

PRAGMA

A publication of Semaphore Corporation

Issue Number 1

August, 1982

IN THIS ISSUE

ZIP Code File Design.....	4
Deriving Year-to-Date Counts.....	14
Vanilla, Part 1: The Parts File.....	15
Sysmap, Part 1: A Cross-Reference System.....	22
How to Flag Missing Checks.....	26
Computing Modulos with ENGLISH.....	29
Making Item Identifiers Hash Well.....	33
Patching the Exit Problem in 3.2 SCREENPRO.....	36

DEPARTMENTS

Utilities.....	9	Command Files.....	30
User Profile.....	18	Queries.....	32
Benchmarks.....	21	The Computer Room.....	34
Wish List.....	27	Letters.....	35
Games.....	38		

PRAGMA

Issue Number 1

August, 1982

Pragma is published at least four times a year by

Semaphore Corporation
207 Granada Drive
Aptos, California 95003

Entire contents copyright © 1982 by **Semaphore Corporation**. All rights reserved. No part of this journal may be reproduced, transmitted, transcribed, stored in a recording, retrieval or computer system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of **Semaphore Corporation**.

Semaphore Corporation offers no warranty, either expressed or implied, for any losses due to the use of any material published in **Pragma**.

SUBSCRIPTIONS

Subscriptions are \$25 per issue, \$100 per year (four issues) in the USA, \$38 per issue to other countries by airmail. All payments must be in US dollars drawn on a US bank.

Paid subscribers who refer new subscribers will receive a free subscription extension of one issue for each referral. Such referrals must include an appropriate subscriber number as explained on subscription order forms.

Address all subscription correspondence to the **Pragma** Circulation Manager, **Semaphore Corporation**. When writing, enclose your issue's mailing address label or the numbers from the upper right corner of your label.

Address changes should be sent at least four weeks in advance. Include old and new addresses along with your issue's mailing address label.

SUBMITTALS

All correspondence and material received will be considered for publication. Except for correspondence used in the Letters Department, authors are paid up to \$200 per full published page for submitted material used in **Pragma**. Actual payment amounts are decided by the Editors and vary with the amount of required editing and rework and with the length of each submittal. Authors are also granted free subscription extensions of one issue for submittals of at least one full published page. Address all submittals and correspondence to the **Pragma** Editors, **Semaphore Corporation**. All letters to the Editors are welcome and as many as possible will be published in the Letters Department.

All submittals must be typed on white paper and double spaced. The first page of any submittal and any accompanying material must include the author's name, address, telephone number and the date. All pages must be numbered.

Hand-typed program listings and other simulated printouts will not be accepted. Authors should submit actual computer-generated output. Print all listings with a fresh black ribbon on continuous white paper. Do not print on perforations. Do not include page numbers or headings on listings in order to simplify reduction and layout. Accompanying documentation should refer to listings by content, line number or symbolic labels, not by page number.

Drawings, schematics and other illustrations must be in black ink on white paper and drawn in a large scale to allow significant reduction. Photographs must be black and white glossies.

Manuscripts are submitted at the author's risk. Unused manuscripts will be returned if a stamped, self-addressed envelope is included. Requests to review galleys must accompany the manuscript when it is first submitted. The Editors reserve the right to edit all submittals.

ADVERTISING

Send all advertising correspondence, requests for advertising rates, and advertising copy to the **Pragma** Advertising Manager, **Semaphore Corporation**. Advertisers sending press releases are requested to telephone the Advertising Manager for an interview at 408-688-9200 within two weeks after submitting the material.

READER SERVICES

Is there a service **Semaphore Corporation** can provide for you? Address all inquiries to **Pragma** Reader Service, **Semaphore Corporation**, or telephone 408-688-9200.

TRADEMARKS

Ultimate TM of Ultimate Corp. • Mentor TM of Applied Digital Data Systems • Evolution TM of Evolution Corp. • Information TM of Prime Inc. • Reality TM, Royale TM, DATA/BASIC TM, ENGLISH TM, RUNOFF TM, SCREENPRO TM of Microdata Corp.

Welcome to Pragma

prag•ma•tism/prag-mə-,tīz-əm/: a practical approach to problems and affairs.

You are now reading the premiere issue of **Pragma** — a journal designed for users of Microdata, Ultimate, Evolution, Prime Information, ADDS Mentor and other “Pick system” computers. As these systems have grown in popularity and stature, certain needs have also developed for users of these systems: the need for information about systems and applications software concepts and techniques, the need for exposure to other users and vendors, the need for a forum for the exchange of ideas, the need for timely data on hardware and software developments. Now **Pragma** is here to help fulfill those needs.

Yes, **Pragma** is intended for Pick system users. But what exactly is a Pick system? An easy definition would be to say that a Pick system is any software that originated with Richard Pick and his associates. That certainly enables someone to tell a Pick system apart from some other system, but it still does little to describe what a Pick system is. And it quite wrongly implies that all Pick systems are alike.

What do you get when you get a Pick system? Is it a certain list of features? And what of the variations and enhancements implemented by the various system vendors? What happens when systems maintained by different vendors evolve to the point of having more differences than similarities? Is it still meaningful to continue to describe such systems as “Pick systems”? Is some sort of standard appropriate? Or would a standard do more harm than good?

Esoteric problems for Pick users are actually going to be only a small part of what **Pragma** will be devoted to. The name **Pragma** is designed to reflect the *pragmatic* flavor this journal will so heavily feature. There will be a notable emphasis on software. *Useful* software. A major goal of **Pragma** is to publish as much software as possible. Not just toy programs or examples of some concept or technique, but complete, usable, working systems and applications of interest to a wide variety of computer users. A good selection can already be found in this issue, and much more is planned for the issues to come.

Together, the Pick system users form an unusual group within the computer community. It is rare to see one family of system software (as opposed to one language or one application system) operated by so many users on so many different pieces of hardware. In spite of (or more likely because of) this unusual situation, Pick systems continue to gain wide acceptance. And with this premiere issue, **Pragma** has completed the first step in a plan to be a continuing part of the Pick system scene. We hope you will join us.

□

—The Editors

Pragma Is Looking for Submittals

Do you have an application program you would like to share? Have you discovered a bug in your system software? Do you have a question about your system for which you haven't been able to find an answer? Have you purchased a software product and written an evaluation of it? Have you performed benchmark tests and recorded the results? Perhaps you've written a small utility that has proven useful? Or maybe you've even programmed a game?

If you answered YES to any of the above, then you're on your way to being the author of an article in **Pragma**. Authors are awarded payments of up to \$200 per full published page for submitted material used in **Pragma**. See the details under SUBMITTALS on the opposite page.

Send your material to the **Pragma** Editors, 207 Granada Drive, Aptos, California, 95003.

□

SUBSCRIBE TO PRAGMA NOW

- ☐ 1 YEAR (4 issues) \$100⁰⁰
☐ 1 YEAR (Foreign) \$152⁰⁰
U.S. FUNDS ONLY

☐ YES, you may publish and announce my name as a new subscriber in the next issue.

My subscriber number on this issue's address label is _____.

Name _____
PLEASE PRINT

Company _____

Address _____

City _____ State/Province _____

Country _____ ZIP/Mail Code _____

Telephone _____

I was referred by
paid subscriber
number _____

Please extend their
subscription by one
FREE issue.

Enclose payment and
mail to:

PRAGMA
207 Granada Drive
Aptos, CA 95003

ZIP Code File Design

Software and techniques for handling the input and storage of U.S. ZIP codes are described. Advice is given for extending the methods to additionally allow foreign mail codes.

A business data processing system typically includes a number of files that contain mailing addresses. Sales order files, customer files, employee files, vendor files, check files, purchase order files, invoice files — all are examples of places in a computer system where one might find an address stored. At one typical single-CPU Microdata installation (a Silicon Valley electronics manufacturer doing about \$25 million in business per year), there are over a dozen different files containing address data.

Analysis of such files typically reveals that a large percentage of the data in each item in the file is devoted to mailing address text. In an accounts payable check file, for example, a typical record may be two-thirds accounting data (an account number, an invoice number, a date, an amount, and so on) and one-third mailing data (name, address, city, state, ZIP).

In order to avoid duplication of data throughout such a file, integrated data base systems often use the technique of only storing in each item a key to some other file where the complete address text resides. For example, a check file might only include the number of the vendor being paid, and the vendor master file contains the "pay-to" address for each vendor. The check printing program is then designed to retrieve the address from the vendor master file as each vendor number is found in the check file, as each check is being printed. This way, instead of storing the address data repeatedly in numerous checks on file, the address is stored once in the master file and only retrieved when needed.

There is not always a guarantee that the desired address for a document will be in some master file. Sales orders, for example, usually include two addresses: a customer "bill-to" address and a "ship-to" address. It should be no problem to design a system for handling such sales orders to assume the bill-to address will stay fairly constant, and so can be stored once in the customer master file. Ship-to addresses, on the other hand, may easily end up being different on each sales order, and so will be stored in the sales order file itself.

Regardless of to what degree a system centralizes address data, very often the system neglects to take one simple extra step to condense data and avoid the duplication of addresses through a file: the step of maintaining a ZIP code file. A ZIP code file capitalizes on the fact that a ZIP code represents a city and state, so that once a ZIP code file is established, it is never necessary to store the name of a city and state in any other file. (Although the principles about to be demonstrated apply just as well to the infamous nine-digit ZIP codes, only five-digit ZIPs will be discussed here in order to simplify the examples).

To understand how a ZIP code file is used, let's first look at an application that does not take advantage of a ZIP file. Let's assume we are working with an application that requires an employee master file. (The program and data for this and subsequent examples were developed on a Microdata system.) Figure 1 shows a DATA/BASIC program for updating such a file. Figure 2 shows the display generated by the program using the SCREENPRO item shown in Figure 3. Figure 4 is a sample of actual file items created with the program (the names and numbers are random data generated for testing purposes), and Figure 5 shows selected dictionary items for the file.

We see that the operator maintains eight different data fields, and that there is no guarantee that the city, state and ZIP will be a valid, legible combination. The ZIP field has at least been separated from the city and state data, to allow sorts on the file by ZIP code. Also, the SCREENPRO item forces the


```

453
001 MIKE N
002 TINNER
003 2844 CHRIS CL
004
005 IRIS, CA
006 94473
007 M
008 847-32-4211

```

```

223
001 MARY
002 STEWART
003 43 CYPRESS LN
004
005 LOMPOC, CA
006 94883
007 F
008 883-42-4422

```

```

326
001 JANE L
002 DERLY
003 122 MAPLE ST
004 APT B
005 PEARSON, CA
006 95648
007 F
008 845-49-4416

```

Figure 4: Employee file items.

CITY.ST	ZIP
001 A	001 A
002 5	002 6
003	003
004	004
005	005
006	006
007	007
008	008
009 L	009 L
010 30	010 5

Figure 5: Employee file dictionary items.

GET.EM

```

001
002
003 ENUMJFNAMEJLNAMEJADR1JADR2
JZIPJCITY.STJSEXJSSNJOK
004 Y
005 Employee Number: \\F
first Name and Initial: \\
Last Name: \\
Address Line 1: \\
Address Line 2: \\
City, State: \\
ZIP: \\
Sex: \\
SSN:
006 0,0
007 JLNAME\ADR1\ADR2\CITY.ST\Z
IP\SEX\SSN
008
009 SJJSJSJSJSJSJSJSJSJD
010
011 LJLJLJLJLJLJLJLJLJL
012 1J1J2J3J4J5J6J7J8
013 YJYJYJYJYJYJYJYJY
014
015
016 0,24J2,24J3,24J4,24J5,24J7
,24J6,24J8,24J9,24J22,9
017 10J20J30J30J30J5J30J1J1J1J2
018
019 JJJJJJZIP:X;:1
020 10J20J30J30J30J5J30J1J1J1J2
021 JJJJJJ6
022 JJJJJJJJJFI or EX:
023 JJJJJJJJJ22,0
024 0,24J2,24J3,24J4,24J5,24J7
,24J18,24J19,24J22,9
025 JJJJJJJJJ12
026
027 10
028 JJJJJ(5N)JJ('M')O('F')J(3N
'-'2N'-'4N)
029
030
031 MD0
032 FJJJJJJJJJJ('FI')F\GOK
033
034
035
036
037
038
039
040 ('EX')E

```

Figure 6: SCREENPRO item for employee file program using a translate conversion.

```

CITY.ST
001 A
002 6
003
004
005
006
007
008 TZIP:X::1
009 L
010 30

```

Figure 7: Employee file dictionary item for translating via ZIP file.

ZIP to be five digits, so that letters and other arbitrary characters can't be entered. However, one can still find many installations where city, state and ZIPs in applications programs are all programmed to be input as one field.

The screen definition item in Figure 6 shows a better approach. Now the operator has to only enter the ZIP, and the system automatically displays the city and state, through the use of a translate conversion and a ZIP code file. Data entry for the employee file is now shorter, quicker and less error-prone since the operator is no longer required to provide the city and state. Data items are smaller, since the city and state are maintained in the ZIP file instead of stored (often redundantly) in every employee record. Figure 7 shows the new dictionary item for retrieving cities and states when listing employees, once data is set up in a separate ZIP file.

Complete ZIP file tapes can be acquired from the government, or a program can be written for input of ZIP file items. Figures

```

GET.ZIP
001 OPEN "DF" ELSE STOP "ERR1"
002 READ SCR FROM "#GET.ZIP" ELSE STOP "ERR2"
003 OPEN "ZIP" ELSE STOP "ERR3"
004 100 *
005 INPUT ZNUM USING SCR, "" SETTING NXT.STP ELSE STOP
006 ZNUM = ZNUM<1>
007 READ ZIP FROM ZNUM ELSE ZIP = ""
008 INPUT ZIP USING SCR, ZIP AT NXT.STP ELSE GO TO 100
009 WRITE ZIP ON ZNUM
010 GO TO 100
011 END

```

Figure 8: DATA/BASIC program for updating a ZIP file.

```

GET.ZIP
001
002
003 ZIPJCITY.STJOK
004 Y
005 ZIP Code:\\
City, State:
006 0,0
007
008
009 SJSJD
010
011 LJJLL
012 JJJ
013 YJYJY
014
015
016 0,24J2,24J22,9
017 5J30J2
018
019

```

```

020 5J30J2
021
022 JIFI or EX:
023 JJ22,0
024 0,24J2,24J22,9
025 JJ2
026
027
028 (5N)
029
030
031
032 FJJ('FI')F\GOK
033
034
035
036
037
038
039
040 ('EX')E

```

Figure 9: SCREENPRO item for ZIP file program.


```

026
027 10
028 ]]]]](VZIP)]]('M')O('F')](
    3N'-'2N'-'4N)
029
030

```

Figure 10: Using (VZIP) instead of (5N) as in Figure 6.

```

          94473
001 IRIS, CA

          1152*BELGIUM
001 ROTTS, SYL

          95648
001 PEARSON, CA

          94883
001 LOMPOC, CA

          2060*AUSTRALIA
001 LILYDALE, VA

          V7H3M2*CANADA
001 BLUE LAKE, BC

```

Figure 11: ZIP file items.

```

CITY.ST
001 A
002 0
003
004
005
006
007
008 A:IF 10="" THEN 9(TZIP:X;;
    1) ELSE (9:"*":10)(TZIP:X;
    :1)
009 L
010 30

```

Figure 12: Checking country before translating through ZIP file.

8 and 9 show a program for maintaining a simple ZIP file in the format used by the employee update program. (A better ZIP file format would be to store only the city in attribute one and to store the state in attribute two. Then validation of state names and sorting by state can be done easily.)

The ZIP update program only needs to be used each time a new ZIP code is encountered and not found in the ZIP file. With the code in Figure 6, the operator will notice when a new ZIP has just been input because no city or state will be displayed. That indicates the ZIP should be entered on the system using the ZIP update program. Or, the code can be modified to refuse to continue data entry if "invalid" ZIPs (ZIPs not yet in the ZIP file) are used. Figure 10 shows the portion of the code from Figure 6 changed to include a validation test, which forces the operator to first enter the city and state with the ZIP update program before the ZIP can be used in the employee update program.

If foreign addresses are a requirement, then programs and files must be set up to deal with an additional field for the name of a country. The ZIP file item identifier must be extended to something like CODE*COUNTRY, where *COUNTRY does not appear for the country with the most entries in the file. Figure 11 shows some examples of ZIP file data with foreign entries. Figure 12 shows a typical dictionary definition that translates through the ZIP file for a customer file where the mail code is stored in attribute 9 and the country, if foreign, is in attribute 10. Similarly, input programs must be modified to take the country into account before the city and state (or province) can be displayed.

ZIP files are simple to implement, quick and easy to use, prevent input errors, and save on file storage. Every installation should evaluate including ZIP file processing in their applications systems.

P

Pragma is Preparing an Index

Pragma will be periodically publishing a complete master index for all articles appearing in **Pragma**, as a reference tool for its readers. Articles will be cross-referenced by keywords found in each article's title, by the author's last name (for submittals), by additional miscellaneous keywords for various subject areas, and by department. Also, all advertisements will be indexed by vendor, product name and product type.

P

utilities

RENUMBER: A Program for Renumbering Statement Labels

A program for resequencing the statement labels in other programs is presented. The program inputs source code and outputs equivalent code, except that all statement labels are equally spaced numerically and in ascending order.

Sooner or later, every programmer is stuck with a project requiring the modification, conversion or maintenance of a piece of someone else's software. Sometimes the software will be small, well-written, structured, straight-forward and (in rare cases) *documented*. More often, the code is huge, messy, unstructured, obscure and barely understandable. If the program has had a history of patches and fixes, or has been maintained by more than one programmer, and the programming language only offers numeric labels, then the code probably suffers from "shotgun" statement numbers: the sequence of

Utility programs have saved many a programmer from the less appealing aspects of software development and maintenance. Reformatting code, stripping files clean of control characters, converting data from one format to another — all are examples of tasks best delegated to a program and not a programmer.

Good programmers will collect a "toolbox" of utility software to use from time to time. If you have a useful utility, send it in for publication. A regular feature in *Pragma* will be this Utilities Department, where good software tools will be spotlighted.

statement labels will be something like

100, 26, 910, 911, 21, 25, 360...

When a statement in the middle of the program says GOTO 47, which way does it go?

RENUMBER is a program for cleaning up other programs that suffer from shotgun statement numbers. RENUMBER inputs source code and outputs equivalent code, except that all statement labels are equally spaced numerically and in ascending order.

RENUMBER is written in Microdata DATA/BASIC (see listing on page 10) and is designed to process other source programs written in DATA/BASIC, but since RENUMBER only pays attention to a small subset of the program being scanned, RENUMBER should be able to run on other vendors' systems without many, if any, modifications.

RENUMBER begins execution at line 11, where the input file and the name of the item to be processed are determined. Most users will probably prefer to catalog RENUMBER and modify this part of the code to automatically extract the file and item name from the command line that invokes RENUMBER, so that using RENUMBER becomes similar to using other system commands like COPY or BASIC. The method shown here was used because it was simple, straightforward and did not add complications that might make the beginning of the program more difficult to understand.

The actual source program to be renumbered is retrieved in line 19 and placed in the SOURCE.ITEM variable. When RENUMBER is finished, OBJECT.ITEM (initialized in line 20) will hold the equivalent but renumbered item. The number of lines in the source item is determined in line 21. OLD.LABELS is initialized in line 22. This variable grows to be a list of all statement label numbers found in the SOURCE.ITEM, with line 44 placing each number found into the next attribute in the list. LABEL.COUNT is a count of the statement labels found so far.

Lines 24 to 30 show that RENUMBER makes two complete passes over the input item. When PASS is 1, RENUMBER simply scans the item and finds all statement numbers so they can be saved in OLD.LABELS. When PASS is 2, RENUMBER scans the item again but replaces each occurrence of a statement number with a new number. For each PASS, the loop from lines 25 to 28 is executed, with RENUMBER calling subroutine 10 to do the processing of each line in the input program. On each PASS, an asterisk is output by line 27. Line 29 does a carriage return and line feed between PASSes, halfway through the total number of

asterisks that will appear on the display. When line 31 is reached, RENUMBER will have stored the equivalent renumbered code in OBJECT.ITEM, which is then written to the same input file and item name, but with a question mark prefixed to the item name. (The question mark reflects the doubt that was present the first time RENUMBER was tried. It was felt to be too risky to leave off the question mark and let RENUMBER possibly clobber the original source item with some unpredictable result.)

Subroutine 10 does all the processing for one line of the input program. Line 35 extracts the next line of interest from SOURCE.ITEM, then line 36 resets COLUMN so that subroutine 20 can later dish out characters column by column.

The LOOP from lines 37 to 70 causes the current SOURCE.LINE to be scanned token by token, where a "token" is a number, a symbol (keyword or variable name), a string (characters enclosed in quotes), an end-of-line, or some other syntactic unit (such as a semicolon or an asterisk). The call to subroutine 50 in line 38 gets the next token in the source line. Subroutine 50 (starting on line 92) scans the line character by character, eventually RETURNing with the actual token saved in the TOKEN variable, and with a number from 1 to 5 (see the EQUATE list in lines 5 to 9) stored in TOKEN.TYPE to indicate the kind of thing TOKEN is.

Subroutine 50 begins by skipping any leading blanks (lines 93 to 95). Subroutine 20 is called to get the next input character. If the character is null, subroutine 50 RETURNs in line 97, indicating end-of-line. Otherwise, subroutine 50 builds TOKEN in one of four ways. If line 99 tests true, the character is a letter, so this must be the start of a symbol (RENUMBER assumes the input program is syntactically correct — in other words, has passed through the BASIC compiler with no errors). Line 100 calls subroutine 40 to gather up all subsequent letters or digits to form the symbol in TOKEN. Subroutine 30 is used in line 89 to "back up" after having gone one character too far in gathering the symbol.

If line 102 tests true, then subroutine 50 has found the start of a quoted string, so subroutine 20 is called from line 104 to gather up the whole string and return it as a token. RENUMBER treats strings as a token to avoid being tricked into thinking that words and numbers imbedded in the string are significant.

If line 107 tests true, then a number has been found, and lines 109 to 114 gather up the number as a token in a way similar to subroutine 40. Since line 116 always tests true, any other kind of character found by subroutine 50 will be treated as a one-character token, which ends up being ignored.

Continued on page 14

```

001 EQU ltrs TO "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
002 EQU noltrs TO "0123456789.$"
003 EQU am TO CHAR(254)
004 EQU yes TO 1
005 EQU return TO 1
006 EQU number TO 2
007 EQU symbol TO 3
008 EQU string TO 4
009 EQU other TO 5
010 *
011 PRINT "File name":
012 INPUT FILE.NAME
013 OPEN FILE.NAME ELSE
014   PRINT FILE.NAME:" is not a file name!"
015   STOP
016   END
017 PRINT "Item name":
018 INPUT ITEM.NAME
019 READ SOURCE.ITEM FROM ITEM.NAME ELSE SOURCE.ITEM = ""
020 OBJECT.ITEM = ""
021 LINE.COUNT = COUNT(SOURCE.ITEM, am) + 1
022 OLD.LABELS = ""
023 LABEL.COUNT = 0
024 FOR PASS = 1 TO 2
025   FOR LINE.NUMBER = 1 TO LINE.COUNT
026     GOSUB 10 : * DO.LINE
027     PRINT "*":
028   NEXT LINE.NUMBER
029   PRINT
030 NEXT PASS
031 WRITE OBJECT.ITEM ON "?" : ITEM.NAME
032 STOP
033 *

```

RENUMBER PROGRAM LISTING

RENUMBER LISTING CONTINUED

```
034 10 * DO.LINE:
035 SOURCE.LINE = SOURCE.ITEM<LINE.NUMBER>
036 COLUMN = 1
037 LOOP
038   GOSUB 50 ;* GET.TOKEN
039   IF (TOKEN # "*") AND (TOKEN # "!") AND (TOKEN # "REM") THEN
040     IF TOKEN.TYPE = number THEN
041       BEGIN CASE
042         CASE PASS = 1
043           LABEL.COUNT = LABEL.COUNT+1
044           OLD.LABELS<LABEL.COUNT> = TOKEN
045         CASE PASS = 2
046           GOSUB 60 ;* REPLACE
047         END CASE
048       END
049     LOOP UNTIL (TOKEN.TYPE = return) OR (TOKEN = ";") DO
050     IF PASS = 2 THEN
051       BEGIN CASE
052         CASE TOKEN = "GO"
053           GOSUB 50 ;* GET.TOKEN
054           IF TOKEN = "TO" THEN GOSUB 50 ;* GET.TOKEN
055           GOSUB 70 ;* REP.LIST
056         CASE (TOKEN = "GOSUB") OR (TOKEN = "GOTO")
057           GOSUB 50 ;* GET.TOKEN
058           GOSUB 70 ;* REP.LIST
059         CASE TOKEN = "RETURN"
060           GOSUB 50 ;* GET.TOKEN
061           IF TOKEN = "TO" THEN
062             GOSUB 50 ;* GET.TOKEN
063             IF TOKEN.TYPE = number THEN GOSUB 60 ;* REPLACE
064           END
065         END CASE
066       END
067     GOSUB 50 ;* GET.TOKEN
068   REPEAT
069   END ELSE TOKEN.TYPE = return
070 UNTIL TOKEN.TYPE = return DO REPEAT
071 IF PASS = 2 THEN OBJECT.ITEM<LINE.NUMBER> = SOURCE.LINE
072 RETURN
073 *
074 20 * GET.CHAR:
075 CHAR = SOURCE.LINE[COLUMN,1]
076 COLUMN = COLUMN+1
077 RETURN
078 *
079 30 * CHAR.BACK:
080 COLUMN = COLUMN-1
081 RETURN
082 *
```


RENUMBER LISTING CONTINUED

```
083 40 * GET.SYMBOL:
084 LOOP
085   GOSUB 20 ;* GET.CHAR
086 WHILE (INDEX(1trs,CHAR,1)OR INDEX(no1trs,CHAR,1))AND(CHAR#"")DO
087   TOKEN = TOKEN : CHAR
088 REPEAT
089 GOSUB 30 ;* CHAR.BACK
090 RETURN
091 *
092 50 * GET.TOKEN:
093 LOOP
094   GOSUB 20 ;* GET.CHAR
095 WHILE CHAR = " " DO REPEAT
096 TOKEN = CHAR
097 IF TOKEN = "" THEN TOKEN.TYPE = return ; RETURN
098 BEGIN CASE
099 CASE INDEX(1trs, CHAR, 1)
100   GOSUB 40 ;* GET.SYMBOL
101   TOKEN.TYPE = symbol
102 CASE (CHAR = "'") OR (CHAR = '"')
103   LOOP
104   GOSUB 20 ;* GET.CHAR
105   UNTIL (CHAR = TOKEN) OR (CHAR = "") DO REPEAT
106   TOKEN.TYPE = string
107 CASE CHAR MATCHES "1N"
108   START.COL = COLUMN-1
109   LOOP
110   GOSUB 20 ;* GET.CHAR
111   WHILE CHAR MATCHES "1N" DO
112     TOKEN = TOKEN : CHAR
113   REPEAT
114   GOSUB 30 ;* CHAR.BACK
115   TOKEN.TYPE = number
116 CASE yes
117   TOKEN.TYPE = other
118 END CASE
119 RETURN
120 *
121 60 * REPLACE:
122 LOCATE TOKEN IN OLD.LABELS,1 SETTING POS ELSE
123   PRINT "Label " : TOKEN : " missed on Pass One!"
124   STOP
125   END
126 NEW.LABEL = 10 * POS
127 PAST.COL = START.COL + LEN(TOKEN)
128 LEFT.PART = SOURCE.LINE[1,START.COL-1]
129 RIGHT.PART = SOURCE.LINE[PAST.COL,LEN(SOURCE.LINE)-PAST.COL+1]
130 SOURCE.LINE = LEFT.PART:NEW.LABEL:RIGHT.PART
131 COLUMN = START.COL + LEN(NEW.LABEL)
132 RETURN
133 *
134 70 * REP.LIST:
```

RENUMBER LISTING CONTINUED

```

135 LOOP WHILE TOKEN.TYPE = number DO
136   GOSUB 60 :* REPLACE
137   GOSUB 50 :* GET.TOKEN
138   IF TOKEN = "," THEN GOSUB 50 :* GET.TOKEN
139 REPEAT
140 RETURN
141 *
142 END

```

RENUMBER CROSS REFERENCES

FV	013 019* 031*
10	026 034*
20	074* 085 094 104 110
30	079* 089 114
40	083* 100
50	038 053 054 057 060 062 067 092* 137 138
60	046 063 121* 136
70	055 058 134*
CHAR	075* 086 087 095 096 099 102 105 107 111 112
COLUMN	036* 075 076* 080* 108 131*
FILE.NAME	012* 013 014
ITEM.NAME	018* 019 031
LABEL.COUNT	023* 043* 044
LEFT.PART	128* 130
LINE.COUNT	021* 025
LINE.NUMBER	025* 025* 035 071
NEW.LABEL	126* 130 131
OBJECT.ITEM	020* 031 071*
OLD.LABELS	022* 044* 122
PASS	024* 024* 042 045 050 071
PAST.COL	127* 129
POS	122* 126
RIGHT.PART	129* 130
SOURCE.ITEM	019* 019* 021 035
SOURCE.LINE	035* 071 075 128 129 130*
START.COL	108* 127 128 131
TOKEN	039 044 049 052 054 056 059 061 087* 096* 097 105 112* 122 123 127 138
TOKEN.TYPE	040 049 063 069* 070 097* 101* 106* 115* 117* 135
am	003 021
ltrs	001 086 099
noltrs	002 086
number	006 040 063 115 135
other	009 117
return	005 049 069 070 097
string	008 106
symbol	007 101
yes	004 116

Once subroutine 10 has had subroutine 50 retrieve the first token on the source line, line 39 tests to see if the token is an asterisk, exclamation point or the word "REM", any of which indicate a comment line. If it is a comment, then line 69 makes it appear to be an end-of-line, so the test on line 70 will succeed, the scanning LOOP will exit to line 71, and the program won't have to bother scanning the rest of the comment. If the source line is not a comment line, then we reach line 40, which tests to see if the token is a statement label number (the only kind of number that can start a line of code).

If the token is a number, then (if PASS is 1) the number is added by lines 43 and 44 to the list of statement numbers found so far. If PASS is 2, then subroutine 60 is called at line 46 to replace the number with a new number. Subroutine 60 LOCATEs the statement number in the OLD.LABELS list at line 122. The new statement number is derived by line 126 by taking the label's POSITION in OLD.LABELS and multiplying by ten. This means the new labels generated will be 10, 20, 30 and so on. If line 126 is changed to multiply by a hundred instead of by ten, then the sequence 100, 200, 300...will be used for the new labels. Lines 127 to 131 are the trickiest part of RENUMBER. This code takes SOURCE.LINE and replaces the old statement number with the new number, then readjusts COLUMN so further input continues where it left off. The code makes use of START.COL, which indicates just where in the input line the statement number starts. START.COL is set in line 108 when the statement number is gathered as a token.

Once line 49 has been reached, RENUMBER has finished with the statement's label, if any, and it is time to scan the rest of the source line. If PASS is 2, then lines 51 to 65 are executed to fix up any statement label references, otherwise PASS is 1 and the rest of SOURCE.LINE is ignored by repeated calls to subroutine 50 at line 67.

Since RENUMBER changes every statement number, every reference to a statement number in the input code must be changed, too. Statement number references are only found in DATA/BASIC after the keywords GO, GOTO, GOSUB or TO (as in GO TO or RETURN TO), which may be a list of statement numbers separated by commas (as in ON statements). If RENUMBER finds the TOKEN "GO", then line 52 tests true, and subroutine 50 is called to get the next TOKEN. If the next TOKEN is the optional "TO", subroutine 50 is called again in line 54 to get to the next TOKEN, which should be a statement number. Subroutine 70 is then called to replace all statement numbers that are found in the rest of the input line by alternately calling subroutines 50 and 60 and skipping any commas. Similarly, lines 57 and 58 handle the GOSUB and GOTO (as one word) cases, and lines 60 to 64 handle the RETURN TO case. Line 71 makes sure the converted SOURCE.LINE is placed in OBJECT.ITEM.

RENUMBER executes rather slowly, partly because the input/output is done on a line by line basis and attribute extraction and concatenation is used to read SOURCE.ITEM and to build OBJECT.ITEM. (DATA/BASIC programs suffer for not having available a quick character stream type of input/output so that utility programs written in DATA/BASIC can read and write source and object files as quickly as the system assembler or compiler.) But RENUMBER gets the job done, as can be seen by RENUMBER's own source listing — its statement numbers are perfect because after RENUMBER was written, it was used to renumber itself.

■

IN THE NEXT ISSUE: trim delim

Deriving Year-to-Date Counts

Year-to-date or just to-date? Maintaining a year-to-date total may not be as attractive as simply keeping a running total that is never reset at year end.

Often, a file item will contain an attribute or two dedicated to accumulating totals for some time period. For example, a vendor master file might contain an attribute named YTD.PO.COUNT, which contains the number of purchase orders written for the vendor so far this year, and an attribute named PRIOR.YTD.COUNT, which contains the value that YTD.PO.COUNT reached at the end of the prior year. Typically, an end-of-year cleanup program is available to go through the file and set PRIOR.YTD.COUNT equal to YTD.PO.COUNT, and then clear the YTD.PO.COUNT attribute.

Designers of such software may want to consider a slightly different method of tracking the data: use one attribute named PO.COUNT, which contains the number of purchase orders ever written for the vendor, and an attribute named PRIOR.YR.COUNT, which contains the value that PO.COUNT reached at the end of the prior year. The current year's count can still be derived (by simply subtracting PRIOR.YR.COUNT from PO.COUNT), but the total count of orders ever written to the vendor is also directly available in the PO.COUNT attribute — a bit of data not available using the previously described system. Also, the end-of-year cleanup program is a bit simpler: just set PRIOR.YR.COUNT equal to the PO.COUNT value.

■

VANILLA

The No-Frills Manufacturing System

Part 1: The Parts File

A series of articles on the design and implementation of Vanilla, a software system for manufacturing, is introduced. Software and techniques for the creation and maintenance of parts files is described.

This is the first in a series of articles that will be devoted to the design and implementation of a software system for a hypothetical manufacturing company. The software system will be called Vanilla to reflect its "no frills" approach: Vanilla will concentrate on only providing the bare essentials necessary to support a working manufacturing material requirements planning (MRP) system. Special needs, features, and the kinds of bells and whistles that are often a high-priority software item in real world companies will not be addressed by Vanilla. That is not to say that the software that will be presented here is only good for theoretical discussions and not useable at an actual manufacturing installation. Rather, it is saying that in a typical applications system, 20% of the code does 80% of the work, and so Vanilla is going to be especially concerned with that 20%.

Presenting working applications software is not the only objective, either. Vanilla will serve as an example of only one way to design a system. As various approaches and tradeoffs in designing and implementing Vanilla are discussed, it will become apparent that Vanilla will be more than just sample code — Vanilla will be a vehicle for ideas and experiments in building a significant software system.

The fact that Vanilla will be a manufacturing system is not as limiting as it sounds. The order entry software that will be developed for Vanilla will have to deal with many of the same concepts and techniques that are just as applicable at a distributing company, a job shop or even a retail organization. Each applications area addressed by Vanilla will typically have numerous similarities with the software requirements of other non-manufacturing corporate environments. Vanilla's accounting applications, such as payroll, will be particularly applicable to a wide variety of business organizations. And although Vanilla is being implemented for a hypothetical company, existing corporations will find many parallels between their business and the one run by the imagined owners of Vanilla.

In the past few years, MRP has become big business in the software world. While there are now many "MRP systems" on the market (which imply that being able to compute and output an MRP report is the ultimate goal for any manufacturing software), MRP will play a somewhat less domineering role in Vanilla. From a functional point of view, being able to compute and print the classic MRP report is just one of the many requirements Vanilla will be designed to fill. Instead, Vanilla's main goal is to be *a completely integrated data processing system reflecting all of the major business operations of a company* (a company which happens to be a manufacturing organization). MRP is just a convenient way and a nice excuse for demanding that Vanilla be an integrated and interwoven set of applications that all work together as one efficient system.

Vanilla will be implemented on a Microdata and make extensive use of DATA/BASIC, ENGLISH, SCREENPRO and PROC. Out of all the Microdata system software, SCREENPRO is the least likely to be found on other so-called "compatible Pick systems". The DATA/BASIC code that will be presented here could do all prompting, input and validation with only DATA/BASIC statements, and as a piece of system software, SCREENPRO is weak in a number of areas (handling of multivalues is probably the most notable), but SCREENPRO will nevertheless be used for a number of reasons:

- SCREENPRO allows a drastic reduction in the amount of DATA/BASIC code that works in conjunction with

SCREENPRO screen definition items. This results in source code that's easier to maintain, and smaller object code that is more efficient to run. SCREENPRO executes quite fast, too.

- SCREENPRO is a nice mechanism to insure that the code for data entry functions is of the same consistent style, from program to program. The code defining how input will be handled is essentially reduced to a compact set of tables, instead of line after line of DATA/BASIC code.
- Taking care of many of the details of creating input programs, such as placement of prompts and cursor positioning, is quicker and easier to define and change with a utility like SCREENPRO.
- The trend in database applications is to use tools like SCREENPRO to do as much of the cumbersome work as possible and to avoid custom programming. If there is a

quick and easy way to generate programs, it should be used.

- In the same way ENGLISH can handle many of the requirements for certain types of output, SCREENPRO and similar packages are designed for handling the input side. Experience with SCREENPRO brings an installation one step closer to the state of the art in how to do database input processing.
- For a project like Vanilla, SCREENPRO conveniently isolates many of the distracting details of the system, so that the solutions presented can focus on the problems of the application, not on the problems posed by the system on which the application is being implemented.

And as a final consolation, Vanilla's use of SCREENPRO simply means that Vanilla's DATA/BASIC programs are go-

```

GET.PM
001 OPEN "DF" ELSE STOP "VAN1"
002 READ SCR FROM "#GET.PM" ELSE STOP "VAN2"
003 OPEN "PM" ELSE STOP "VAN3"
004 100 *
005 INPUT PM.ID USING SCR, "" SETTING NEXT.STEP ELSE STOP
006 PM.ID = PM.ID<1>
007 READ PM FROM PM.ID ELSE PM = ""
008 INPUT PM USING SCR, PM AT NEXT.STEP ELSE GO TO 100
009 WRITE PM ON PM.ID
010 GO TO 100
011 END

```

GET.PM	022 Part number (up to 7 digit
001	s):JRevision (any 1 charac
002	ter):JDescription (any 30
003 PARTJREVJDESCJUMJOK	characters):JUnit of measu
004 Y	re (any 2 letters):JFile (
005 \\ Part Number:\\	FI) or exit (EX):
Revision:\\ Descripti	023 22,0J22,0J22,0J22,0J22,0
on:\\Unit of Measure:	024 22,29J22,27J22,32J22,32J22
006 0,0	,23
007 JDESC\UM	025 1J1J1J1J1
008	026 @J@J@J@J@
009 SJJSJSJSD	027 7J2J30J2J2
010	028 J(1X)JJ(2A)J('FI')
011 LJLJLJLJL	029
012 1J1J2J3	030
013 YJJYJJY	031 MD0
014	032 FJJJJF
015	033
016 3,17J5,17J7,17J9,17J22,23	034
017 1J1J1J1J1	035
018 @JJ@J@J@	036
019	037
020 7J1J30J2J2	038
021	039
	040 ('EX')E

Figure 1: Parts file input program.

ing to rely on a single INPUT USING statement to input one or more fields instead of a group of DATA/BASIC statements (PRINT statements to do the prompting, INPUT statements to get the data, and IF statements to do the validation). An INPUT USING statement invoking SCREENPRO can always be thought of as a subroutine call to a group of statements that do the input for a field in whatever fashion is most comfortable to the programmer.

The Parts File

Manufacturers take parts and assemble them to build other parts. Any study of a manufacturer quickly reveals that the parts file is a collection of information of major importance to just about every business function performed throughout a manufacturing company, and so the creation and maintenance of a parts file will be the first issue to be addressed by Vanilla.

The parts file usually comes under the domain of the company's Engineering department. A careful analysis of the way a company engineers its products can uncover a surprising number of factors that must be correctly handled by a computer in order to correctly implement a data processing system that manipulates a parts file. Among these factors are:

PART NUMBER FORMAT. Is the part number of fixed size or variable length? What kinds of characters can appear in a part number? Do the part numbers have any special significance in any of the characters? Or is the number for any next part simply sequentially generated? *In Vanilla, a part number will be any number, up to seven digits long.*

REVISIONS. How are changes to a part's form, fit and function denoted via revision levels? Are revisions alpha, numeric or alphanumeric codes? Does a part and its drawing have separate revisions, which may not happen to coincide? How are preproduction parts handled? Is only the latest version of a part kept on file? What mechanism changes the revision in a file? *In Vanilla, only the latest revision of a part will be on file. The revision is any one optional character, which is controlled by manual updates. The revision indicates the change level for the part, its drawing, and other Engineering documentation, which is always at the same revision level.*

DESCRIPTIONS. What is the maximum description length? MRP doesn't really care about part descriptions, but a parts system is too confusing to humans if descriptions aren't readily available. *In Vanilla, descriptions can be up to any 30 characters.*

UNIT OF MEASURE. Are parts documented, purchased, received, stocked, kitted and built with consistent units of measure? *In Vanilla, a part is documented, stocked, kitted and built all with the same unit of measure, defined by Engineering for the part as any two letters. Purchasing and receiving together may use some other unit of measure.*

OTHER PART ATTRIBUTES. These are typically things like vendor and manufacturer part numbers, document page sizes, and other bits of information of interest to Engineering personnel. (Inventory attributes, such as ABC codes and lot sizes will be considered the domain of inventory personnel, not Engineering, and so will be discussed in a future article. Inventory data is probably better off in separate files anyway.) *For now, Vanilla won't bother with other part attributes.*

Figure 1 is the Vanilla program for part data entry. Since part records are so small, all Engineering data is being kept in one file. For more comprehensive systems where the part record includes dozens of attributes, it is probably better to keep the part's description and revision information in a separate file, since a gigantic percentage of the accesses to the parts file will typically be for just those two pieces of data.

In future installments, a number of Vanilla programs and procs will be presented, in order to complete the Engineering applications for the Vanilla system. By then, standards for user procedures, code format and system and user documentation will also be presented and established, and those standards will be used for the remainder of the Vanilla development effort.

■

IN THE NEXT ISSUE: bills of material

user profile

In Sunnyvale, California, in the heart of Silicon Valley, sits a small company of about 80 employees: Molec-tron, a manufacturer of lasers and pyroelectric detectors. For this issue's user profile, Pragma interviewed Judy Rands, Molec-tron's Information Systems Manager.

Pragma: Molec-tron is a Microdata installation. How were you involved in the acquisition of your Microdata?

Rands: We got it in the fall of 1976. We have a real old machine, serial number 726. The controller at the time sent out requests for proposals. Based on those responses, he chose Microdata. He wrote up all the requirements for what we would need in our data base. The request for proposal was not only just for the hardware, but also for the software that would go with it. He made up samples of the reports, what they should look like. It was a huge job. My input was after they had already picked Microdata. Then it was a question of whether or not we were going to buy packaged programs or

Are all data processing installations the same? How do managers actually manage, how do programmers program, how do operators operate, how do users use their data processing systems? What defines the leading edge of current, modern information processing?

In this regular department, Pragma will be interviewing personnel at a variety of installations, to reveal the who, what, when, where, why and how of actual data processing organizations.

have it done on a custom basis. I thought we should have gotten packaged.

Pragma: What was your reason for preferring a package?

Rands: Because I thought that for the kind of things we wanted, which was standard business things, it was better to have something that a team of people worked on for years and years, rather than try to start from scratch all over again with just the consultants, who may or may not have been accounting experts. Molec-tron was looking at Infoflo. I looked at it and studied it. While there were a few things that didn't really fit with what we were doing, it would have been much easier for us to change our way of doing things, like the length of the numbers or whatever, than it would be to redesign and start all over, I thought.

Pragma: What happened?

Rands: We hired independent consultants. Our big mistake was that, with having these different people do these things, the systems don't fit together like they should. I think everything ought to feed into the general ledger. The reason I got involved was because some of the things that they were putting on the system were things that I had been doing manually.

Pragma: All the software development was done by outside consultants?

Rands: Right. And the problem was that it then fell to the person in charge of each department to test what the consultants had done, and they didn't test it very well. Some of the other systems that I wasn't involved in didn't get tested, and there were so many errors that I inherited the testing to make them right. That's how I got exposed to more and more. The consultants should have tested their stuff, but if they didn't, then the person who was responsible for that area should have, and they couldn't do it either, so then it fell to me to do it. By the time the consultants finished their job, I had learned how to program. The vice president of manufacturing at the time learned how to program and he started programming the manufacturing side of it. He did that for several years, and then he left. Now he still consults with us, and maintains all of his programs.

Pragma: What applications do you have running?

Rands: We do payroll, order entry, invoicing, accounts payable, purchase orders, receiving, stock room, finished goods and, with that, job costing. Personnel stuff, cash receipts. We have general ledger, and from that comes all the financial statements and balance sheets. We collect costs by product code, and we do forecasting, budgeting. There's really not any department that's not affected.

Pragma: Do you have MRP?

Rands: Not really. At any time we can find out what is on order, but we don't project into the future to determine how much we're going to have to order. It's just: what do we need right now. If we set up this job for such and such a quantity, how much would we have to order.

Pragma: Do you plan to do MRP?

Rands: Yes, but we haven't got the requirements that *we* need yet in order to get that, and that is good forecasting. The way the market is, it changes from day to day, and if you don't know what you're going to have to be building in the next few months, there's no point in having MRP.

Pragma: Do you have a well-integrated system that avoids duplication of effort?

Rands: Pretty good, but not totally. If we had planned from the general ledger on down, it would have been much better. But now, as long as it's there in the data base, I can get it for them.

Pragma: How is your backlog of data processing projects prioritized?

Rands: When people come to me with requests, I write them up. I have a folder of all such requests, all future projects, and I go through them and prioritize them. The easy things I do first. I don't refuse to do anything that's a valid request.

Pragma: You have thirteen terminals, but only two are Prisms.

Rands: We have one Hazeltine that has had the board replaced in it three times now, but it's under warranty. You just have to call an 800 number, and they send a new board and install it, and you send the old one back in the same box. So it hasn't been real reliable, but it hasn't cost us anything. Then all the rest of them are ADDS terminals. We've got 580s and a Regent 100 and four Viewpoints. The Viewpoints are fairly new, and we don't have maintenance contracts on any of them. I think that I've only had to send the non-Viewpoint ones back twice, and then we paid time and materials which has been \$100 or less. We have had to send one Viewpoint back. We haven't had much trouble with terminals.

Pragma: How do you plan to upgrade when the time comes?

Rands: Well, I think the main thing I'm going to try and do now is make sure that everything is portable to any Pick operating system. When it comes time then to junk this machine and buy a new one, we'll just have to look and see what the best deal is for any of the Pick operating systems.

Pragma: How will you know when that time has come?

Rands: When we don't have enough disk space. I don't think we're going to need more terminals. If we're going to continue to put on new applications, then this 30 megabytes of disk isn't going to be enough. At that point we're going to have to either buy a 50 megabyte disk, which wouldn't give us much more for the cost of it, or get a new machine.

Pragma: How do you handle your own personal training?

Rands: I read everything I can get ahold of about Pick operating systems. I haven't tried to train myself in any other kind of operating system at all. When we did this upgrade from 2.5 to 3.2, I had read all the manuals before we did that, so I knew about all the changes. I refer to the manuals all the time. That Datastream thing has a lot of user input, how they do things and so forth. So I always read those, and try to incorporate the ideas that I can. I don't think I have very many gaps in my knowledge at this point — of what the machine can do and what it can't do. I have spent a lot of time in the past to

try to work things out on my own, and I've written things up, and I've done a lot of study on it. I don't know the user modes, but anything else I know.

Pragma: Do you feel you yourself are doing an effective job at Moletron?

Rands: I guess I really feel like there's too much for me to do by myself, and yet there's no one else who knows the data base that is in a position to make any kind of decisions or anything. For instance, if somebody has a job that they want to have done, they can dream up all kinds of ways of what should be done... in fact, what I'm working on right now is something where they decided they needed a new input program and they were going to put in all this stuff. Well, it's already there in the data base, and so it's just a matter of getting the information. If they'd gone ahead and asked the other programmer to do it, it would have been a lot of wasted work and duplication. Sometimes they'll come up and want something that's a very easy thing and not any trouble at all to get, but they didn't know that or they would have asked for it a long time before. People don't understand that you can pull stuff from other files, and things like that, and even though you tell them that, it doesn't carry over into the next thing that they want.

Pragma: How satisfied are you with your system documentation?

Rands: Not at all, but we're working on it now. I think the most important thing is to have programmer's documentation, and that's what we're working on. Because you have to know all the updates to a particular attribute, and all the programs that use it, and if you don't know that, then it's pretty hard to find out. That is, as far as I'm concerned, the number one priority. As far as user's documentation, we don't have any. What people have always done is just taught the next person orally and that really seems to work OK. They all have their own little notebooks. No one has ever complained about not having proper instructions. You know it's pretty self-instructive anyway, because the prompts lead them through it. In every department, someone, at some time, has written up the steps to do to go through to get something done. I have tried to conduct classes in the past for operators to do ENGLISH sentences, because that was one of the selling points for Microdata, but in practice it just requires too much knowledge, and operators don't pick that up. They have to understand what an attribute name is, and all that kind of stuff, and they don't understand and they don't have the time to get into it.

Pragma: How's your system's security?

Rands: We do a file save every night, and a restore every Friday night. We keep Friday's tape for the month. You know, Friday 1, Friday 2, and so forth. We don't write over Friday's tape every week, so that we not only can go back 24 hours, but we can go back to any Friday, for four weeks prior to that. We always have a set of tapes offsite, for two weeks back. I have the updates, locks and retrievals on some of the files, so that if you're not logged onto the right account, you can't access the files. We also have an authorization code that's in the proc or in the program that's a control character, so that in order to run the program that updates stockroom inventory, you have to have the control character in the program. So it's pretty good. The payroll stuff is off in a file that nobody would know the name of.

Pragma: What about your file modulus?

Rands: I have a dictionary word in the STAT-FILE called STARS that alerts you if a file is sized wrong. One star means that it should be looked at, two stars means it needs to be changed, three means it for sure needs to be changed. And that works either if the file size is too big or if there's a lot of

overflow. So we just look at that and keep an eye on it. The files are all sized very well.

Pragma: How about system performance?

Rands: Well, it's improved a lot since we went to 3.2 from 2.5, but we also increased from 64K to 96K. We plan to go to 4.1 as soon as possible, but I want to wait until we get a tape that has all the patch reports in it. When they gave us the 3.2 tape, which had been out for years, we got one without any patches. I had to put the patches in myself, and I didn't think the instructions were good.

Pragma: Data processing people are notorious for underestimating projects. By how much time do you think you underestimate your jobs?

Rands: Probably by about 25%. Between 25 and 50%. I'll tell you why that happens, though. It's because of the interruptions I have. I'm not saying that if I could work on the job full time like I would like to, I could get it done in the amount of time that I had allotted. But there're too many interruptions, because I put out fires all day. Sometimes a program that has been running for years has some kind of esoteric bug in it that never showed up before, so then maybe I have to fix the data base, or change a program, and that takes me away from what I thought I was going to be doing. You have to understand that I don't just do things that are computer related. I would say that 50% of my time is spent on non-computer related things. For instance, personnel things. There's a lot of government reports that I'm responsible for, administrative, meetings, that kind of stuff. I input the budget for the whole company. I spend a lot of time doing that sort of thing. I'd say that another 10% is probably ENGLISH reports and that kind of thing. Maybe 10% supervising, or 15% with the new programmer. I don't spend a lot of time programming. It kind of comes in spurts, too. I will shut everything else aside for awhile and try to do nothing but program for a week.

Pragma: Molelectron is getting a word processor. Is the Microdata used for word processing?

Rands: I use it all the time, and I had an opportunity to give input. I thought we ought to get a Microdata word processor, and I couldn't sell it to anybody. We're going to end up taking a lot of customer lists from the computer and typing them into the new word processor. I don't think the people believed that a Microdata was going to be as sophisticated as a real word processor — that it was going to have the kinds of options that they wanted. They just didn't know enough about it, and they were not willing to spend the time to look at the Microdata close enough to see that we could do those things and probably do them better. With the word processor, you have to buy an extra support package, and you can sort on maybe two different attributes, and that was a big selling point. Well, we already had that. The other thing is that the terminals from Microdata could be used either as input terminals or word processing terminals if you switch the keyboards. But then nobody went for it. There's a lot of people who don't like Microdata very well. We went for such a long time with just 64K of core, and it was too slow and there was a lot of complaining, so people don't like it real well. Management doesn't like it. They're not users, but they hear people complaining about how slow it is. And now it's not that slow, and I kept telling them that as soon as we upgrade it's going to be faster, but it's like "we'll believe it when we see it". In the meantime, they got the word processor. I just don't think that it makes any difference to the employees whether it's Microdata or not. They don't realize, since they don't know anything about minicomputers, how neat some of the capabilities are. They don't know the difference between an ENGLISH proc and a proc that runs a program. Maybe they've been from somewhere where everything has to be a program. So to them

its the same, not necessarily any better. The people who appreciate the Microdata are probably the programmers. And since there aren't that many of us here, nobody's particularly loyal to Microdata.

Pragma: Do you think there is some effective way to educate users?

Rands: I really believe that the people who are interested enough to want to learn it are few and far between. I guess I'm now at the place where I wait for someone to come and ask me before I try to train anybody. I've found that the biggest stumbling block is people can't grasp the idea of a unique identifier for each item in a file, and that having different values associated with it, you can get anywhere. I don't know if I wasn't explaining it right or what, but people could not get that concept. To them it's all just a bunch of stuff in there loose, I guess. Say you get a different report from the same file, and it's got different information but maybe there's one thing in common, people don't understand that if you change that piece of data, it's going to be changed on both reports. They think that it's kind of "typed in" there. They really don't understand the idea of sharing the data base. People aren't curious enough to care about that anyway. I'm not saying that they should be curious, I'm just saying that's not where their interests lie. So we don't have many operators that can do anything.

Pragma: You've joined the local user group?

Rands: I find that very beneficial. I would say that they're really a valuable source of information.

Pragma: What about the larger national organizations?

Rands: I haven't ever been to any of their conferences. I'm not a member of any of them. I don't think that Molelectron would pay for me to go. I did buy a tape from one of them, and I knew all the stuff already, so I don't think they're very beneficial. They have seminars, but the stuff is pretty basic. I think the value of the conferences is meeting the people that you do. You learn a lot from them.

Pragma: Overall, how well does Molelectron do data processing?

Rands: I think that we've done a lot on the computer and we're successful on all systems, and because of that we get a lot more information than we could possibly ever get with a manual system. I would never want to give it up and go back to manual. People are accustomed now to getting all this information at their fingertips. The drawbacks that we've had are that, although we're very successful with all of our systems, they don't fit together quite as well as they could. And we're really lacking in documentation. We haven't got enough people who understand it, so that we can back each other up. Those are the drawbacks. But we've saved a lot of money, like not having the bank do our payroll and that kind of thing, so it's been nothing but advantageous to us to have the computer. Since I'm a perfectionist, it's not ideal, but it's not bad, either. I know there are companies that have had them for years and can't do anything — still don't have inventory up or bill of materials up or anything like that. We've had a lot of good use of it for a long time. And I think people like being able to work on it. I think they feel like they're growing if they can learn how to input to a computer, and it helps their job prospects. So I'm really sold on it.

Pragma: Where's Molelectron data processing on a scale of 1 to 10?

Rands: Oh, I think you'd have to say a 10. I don't know of anybody else who's planned it right and bought a package and have everything that feeds into it. I haven't even seen any systems that are designed better, I don't think.

☐

benchmarks

Are you thinking of doing an upgrade to your hardware or software? Are you comparing throughput and performance while shopping around for a system? Have you converted from one machine to another? Be sure to send in your benchmark statistics to Pragma, so the results can be featured in this department and shared with other installations.

More Memory, Fewer Reads

A general improvement in throughput, as indicated by fewer disk reads during program execution, is evident in statistics gathered before and after installation of additional memory.

A Microdata 6000 was recently upgraded from 48K of core memory to 128K. As anticipated, a noticeable improvement in response time became evident — users began to frequently mention “the system is much faster now.”

In order to more accurately gauge the changes in program performance allowed by the additional memory, statistics were gathered both before and after the upgrade (see Figure 1). The numbers were obtained through use of the CHARGES verb.

It is interesting to note that all reported CPU execution times *increased* (for reasons that are not clear), while reported disk reads usually dropped.

The TIMESLICE used was 25 milliseconds in all cases, with only one program running in the system at a time, and with the system printer offline during all tests.

□

	CPU time with 48K	CPU time with 128K	Disk reads with 48K	Disk reads with 128K	% Fewer disk reads
SORT 1:	621.296	632.576	3,527	3,215	9
SORT 2:	118.124	120.021	1,050	888	15
SORT 3:	781.919	783.302	12,333	10,642	14
SORT 4:	182.611	185.715	441	478	-8
SORT 5:	73.066	74.387	387	174	55
SORT 6:	416.616	425.488	1,362	1,129	17

Figure 1: Changes in program execution as reported by the CHARGES verb.

SYSMAP

Part 1: A Cross- Reference System

A series of articles on the use of cross-reference files is introduced. Why cross-references are necessary and what kinds of information must be maintained are discussed.

Except for the most trivial of software, the interactions between an application's programs and data quickly become so complex that it is impossible to keep track of all the programmed interdependencies in a computer system without resorting to written documentation. Of the various types of documentation that a given system might offer, the most important to the programmer are cross-references: documentation that identifies and explains all the interactions that take place in a complicated software system. Programs read and write files, files point to other files, procs invoke programs and other procs, or the procs might even read and write files directly. And through it all, the programmer is expected to identify and understand every one of these interactions and the consequences of modifying any of them, regardless of the total number of programs, procs and files.

SYSMAP is a set of programs and standards for "mapping a system." In other words, SYSMAP is a tool for documenting software. This article is the first in a series that introduces SYSMAP and shows how it can be used to document all the complex interactions that occur in an integrated data base system. To date, SYSMAP has only been implemented on Microdata hardware, and so the examples that will be presented here all have a definite Microdata flavor, but the concepts and techniques used are general enough to be useful on other similar computer systems.

System cross-references are most often used to determine what parts of the system might be affected by any proposed change, or to determine what parts of the system rely on some certain program or piece of data elsewhere in the system. To be comprehensive enough so that it is easy to trace all the possible threads that are woven throughout a large collection of software, a cross-reference system needs to document a number of different relations for the programmer. Specifically, system cross-references must identify:

1. **Which procs invoke which programs.** Procs can start programs executing through the use of the RUN verb, or by referencing the name of a cataloged program as though the name were a verb.
2. **Which procs invoke other procs.** Procs can transfer control to another proc by enclosing a proc name in parentheses, or a proc can call another proc by using square brackets.
3. **Which files and dictionary words are referenced by which procs.** Procs that result in the execution of a TCL command are usually equivalent to a verb referencing a file name and a list of dictionary words.
4. **Which file attributes are read or written by which programs, and which attributes are referenced by which dictionary words.** The various forms of DATA/BASIC READ, WRITE and DELETE statements have the effect of creating or changing one or more attributes in a file item. File attributes are also referenced by dictionary words.
5. **Which file attributes are read via translate conversions, by which programs, procs, screens or dictionary words.** Translate conversions are very powerful tools that can be imbedded just about anywhere, which means they can be overlooked quite easily. A translate conversion might appear in a DATA/BASIC ICONV or OCONV function call, in a proc IH or IBH command, in a SCREENPRO ICONV, OCONV or validate parameter, or in dictionary words at V/CONV and V/CORR.
6. **What the structure and content of each file attribute is.** Knowing only what attributes in a file are being referenced


```

File Attribute. AMC * Description.....
FF          File formats. Describes the contents of all
FF FILE.ATR  0   File name:"*":attribute name. When
                "*" attribute name is not included, then AMC
                and AVS below are null, and DESC describes
                the whole file, not just one attribute.
FF AMC       1 A Attribute mark count (attribute number where
                attribute's data is stored). Only present if
                FILE.ATR includes an attribute name.
FF AVS       2 A Code character indicatins data resolution.
                A = attribute
                V = values
                S = subvalues
                Only present if AMC is 1 or greater.
FF DESC      3 V Description of attribute.
***
XB          Cross reference for Data/Basic programs.
XB PROG      0   Program name.
XB PROCS     1 V Procs that execute program.
***
XD          Cross reference for dictionary words.
XD DICT.WORD 0   File name:"*":dictionary word. When the
                "*" dictionary word is not specified, the
                item id is implied.
XD PROCS     1 V Procs that use word.
XD WORDS     2 V Words that use word via A:N().
***
XF          Cross reference for files.
XF FILE.ATR  0   File name:"*":attribute name.
XF RPROC     1 V Programs that read the attribute in order to
                use its contents.
XF UPROC     2 V Programs that update the attribute by
                changing and then writing or deleting the
                attribute.
XF RPROC     3 V Dictionary words that access the given
                attribute.
***
XP          Cross reference for procs.
XP PROC      0   Proc name.
XP PROCS     1 V Other procs that execute this proc.
***
XT          Cross reference for translate conversions.
XT FILE.ATR  0   File name:"*":attribute name.
XT PROGS     1 V Programs that translate the attribute.
XT PROCS     2 V Procs that translate (any form of read or
                write) the attribute.
XT SCREENS   3 V Screens that translate the attribute.
XT DICT.WORDS 4 V File:"*":word that translates the attribute.
***

```

©

Figure 1: SYSMAP file formats. Since this data was stored in an FF file, SYSMAP was used to generate this listing.

is usually not enough information. More complete cross-reference documentation includes at least a brief description of what each file attribute is used for, along with an indication of whether the attribute's data appears only as a single attribute, as multiple values, or as sub-multivalues.

If all of the above cross-reference information is accurate and up to date for a given set of programs and files, then it is a very simple matter for anyone to determine how one portion of the software relates to any other portion of the system. Of course, not every software system may rely on the types of program interactions listed above. For example, some installations prohibit the use of translate conversions in DATA/BASIC programs, requiring the use of READ statements instead. In that case, the cross-references do not need to make provisions for identifying which attributes are translated by which programs, since the capability is never used.

The best place to keep and maintain all of this information is in a set of files, which means that the cross-reference tools themselves can be a part of the system being documented. That is the idea behind SYSMAP. SYSMAP consists of a set of files and programs that can be added to any existing software system. SYSMAP files contain information describing all the files in the system being cross-referenced, including the SYSMAP files themselves. The SYSMAP programs are simply data entry programs for keeping the contents of the SYSMAP files up to date. Naturally, the SYSMAP files contain descriptions of which SYSMAP programs affect which SYSMAP files.

Figure 1 describes the format of the six different files in SYSMAP. The FF file, which is a kind of data dictionary, describes the content and structure of all files in the system being cross-referenced. The XB file contains information describing which procs in the system invoke which programs. The XD file contains descriptions of the relations between procs and dictionary words. The XF file documents the file references made by programs and dictionary words. The XP file identifies which procs call other procs. And the XT file describes all uses of translate conversions in a system. Since the descriptions for each of these six files are stored in the FF file itself, SYSMAP was used to generate Figure 1. Note that there are provisions for identifying proc-to-proc and proc-to-program references, but program-to-program references are not included (although they could easily be added, probably as a second attribute in the XB file). This is because SYSMAP was developed on a system where DATA/BASIC CALL statements were not used, and where CHAIN and ENTER statements were purposely avoided, as required by the installation's programming standards.

In the next installment, the SYSMAP program for maintaining the FF file will be introduced. There will be actual code examples to demonstrate under what circumstances it is necessary to add information to the other SYSMAP files, and eventually the complete set of all SYSMAP programs for maintaining every SYSMAP file will be presented.

©

CARGO

A Spread Sheet Generator for Microdata Computers!

- MARKET SHARE ANALYSIS
- CAPITAL BUDGETING
- DISCOUNT SCHEDULES
- CASHFLOW
- COMMISSION PLANS
- PRODUCT PRICING
- LONG RANGE PLANNING
- DIVISION SUMMARIES
- CAPACITY PLANNING
- INVENTORY FORECASTS
- PRO FORMAS
- JOB COST ESTIMATING
- INCOME STATEMENTS
- SALES FORECASTS
- "WHAT IF" MODELING
- BALANCE SHEETS
- MAKE OR BUY OR LEASE
- PROFIT & LOSS STATEMENTS
- REQUIREMENTS PLANNING
- CONSOLIDATIONS
- ANY SPREAD SHEET APPLICATION

What is Cargo?

Cargo is powerful software — a program — for Microdata computers that allows users with no computer training to quickly and easily create, compute, manipulate, display and print spread sheets...spread sheets such as profit and loss statements, sales forecasts, budgets — any of the kinds of tabular reports found throughout a business organization. If you are working with columns and rows of numbers and text, Cargo will automate your work — without the help of programmers or other data processing personnel!

Why Use Cargo?

Cargo makes spread sheet revisions and turnaround extremely quick and easy • Cargo avoids programming, especially those expensive program development and maintenance costs • Cargo allows non-programmers to implement useful business applications on the computer • Cargo quickly pays for itself • If desired, Cargo can be incorporated into ongoing development work by programmers, allowing current programming projects to be completed sooner • Cargo output is self-documenting • Cargo completely replaces the necessary programming in most spread sheet applications.

How Does Cargo Work?

To create a spread sheet with Cargo, a user begins by typing into the computer the list of column and row names that border the sheet. Typically, the columns are time periods such as JAN, FEB, MAR...while the rows are categories such as INCOME, SALES, INVENTORY, and so on. Next, the user inputs the starting values for various positions on the sheet. Then, using a simple, menu-driven, "fill-in-the-blanks" approach, the user defines the rules by which the rest of the sheet values should be computed. For example, *Add all rows from row FOOD through row SHELTER put result in row EXPENSES*. At this point, all necessary data for the sheet has been supplied. Cargo always has a current version of the sheet on file, so the user never has to repeat any of the work done to develop a spread sheet. The user can now command Cargo to calculate the remaining values on the sheet and output the final results. If desired, the user may specify parameters to cause the final report to appear in any desired format. Often, the user will want to modify some of the various columns, rows, values or calculation rules and then recompute the complete sheet. This can quickly and easily be done any number of times with Cargo, allowing the user to experiment with report formats, test assumptions about initial or final data values, ask "what if..." questions for planning — in short, use a powerful tool to communicate results, examine alternatives, and improve decision-making.

Features

Designed and offered exclusively for Microdata systems • Totally on-line and interactive • Fully menu driven with continuous prompting • Simple to understand • Easy to use • No prior computer knowledge required • Field proven • No special syntax or formulas to memorize • Complete documentation, all on-line and immediately accessible • Includes step-by-step demonstration examples • Installs in seconds • One copy of Cargo simultaneously supports any number of terminals, accounts and sheets • Absolutely no assembly language code, special user modes, or hardware modifications for insured compatibility with future operating system releases • Fully symbolic: names instead of numbers reference sheet columns and rows (*Add WAGES with DIVIDENDS put result in INCOME* instead of cryptic methods like $ROW\ 3 = ROW\ 1 + ROW\ 2$) • Full complement of sheet calculating operations • Number of columns and rows limited only by Microdata file and record capacities (total column and row name lists may not exceed 32,267 characters) • Cargo has been used for spread sheets with hundreds of rows and columns in each sheet • All data structures transparently maintained as files — simplifies use of Cargo by programmers, yet allows users to totally ignore file system • Defaults used for all parameters to simplify initial sheet creation • Repetitive data easily created by input commands or by calculation • File data can be read into sheets • Multiple sheets easily consolidated • Automatic rotation of time period columns for rolling reports • Sheet copy capability • Portions of sheets can be selected for printing • Calculation results displayable before printing • Sheets can be printed as data capture worksheets, data entry validation sheets, or final report sheets • All calculation rules printable as easy-to-read prose • Printed report widths, spacing, underlining, decimal places, headings, footings and other parameters all under user control.

How To Try Cargo

1. Fill out, sign, and date the License Agreement below. Be sure to include your mailing address, SYSTEM SERIAL NUMBER, and signature. Do NOT use a Post Office box, if possible.
2. Check this box ☐ to order a Cargo magnetic tape for a trial period evaluation. Enclose payment of \$65 to cover media, configuration, handling and shipping costs. California buyers must add 6% sales tax. A complete version of Cargo will be delivered, including all documentation and loading instructions, but the Cargo software will be configured to operate for only 15 days after receipt.
3. Check this box ☐ to order a complete Cargo magnetic tape, including all documentation and loading instructions, for perpetual use. Enclose payment of \$880. California buyers must add 6% sales tax.
4. Please circle your tape drive density: (800 BPI) or (1600 BPI)
and your printer character set: (UPPER & LOWER CASE) or (UPPER CASE ONLY)
5. Send this order, the License Agreement, and your payment to: Semaphore Corporation
207 Granada Drive
Aptos, CA 95003

Your order can be processed faster if the number following your name on the PRAGMA address label is written here:

Cargo is available immediately, but currently only for computers using Microdata operating system release 4.1 or later • Since Cargo is menu driven and uses formatted screens extensively, 9600 baud terminals are recommended (but not required) when using Cargo • Quantity discounts available • Dealer inquiries invited • Telephone Semaphore Corporation at (408) 688-9200.

Non-Exclusive Software License Agreement

Semaphore Corporation will provide a non-exclusive perpetual license to

Company Name _____

Address _____

(hereafter referred to as "Licensee") to use the Cargo software (hereafter referred to as "Software"). The Licensee hereby agrees the Software being licensed, and all alterations, modifications, revisions and enhancements thereto, whether now or hereafter made, constitute proprietary information, rights and trade secrets of Semaphore Corporation, and that title and full ownership rights to the Software and to any alterations, modifications, revisions and enhancements thereto, shall remain in Semaphore Corporation. The Licensee hereby agrees the Software shall be used only by the Licensee on the Licensee's Microdata computer equipment, mainframe serial number _____. The Licensee hereby agrees not to resell, trade or otherwise make the Software available in any form to any other person, persons or company without prior written permission from Semaphore Corporation. The term of this license shall commence upon the signing of this agreement and shall remain in force perpetually unless terminated by Semaphore Corporation. Semaphore Corporation makes no warranty, express or implied, concerning the applicability of this Software to any specific purpose. It is solely the Licensee's responsibility to determine suitability of the Software for a particular purpose. Semaphore Corporation accepts no liability for loss or damage caused, or alleged to be caused, directly or indirectly, by any Software sold by Semaphore Corporation, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such Software. This statement of limited liability is in lieu of all other warranties or guarantees, express or implied, including warranties of merchantability and fitness for a particular purpose.

Signed by _____ Date _____ Telephone _____

Printed Name _____ Title _____

How to Flag Missing Checks

A method for using dictionary words to flag gaps in numeric sequences is presented.

A familiar feature on bank statements for checking accounts is the flagging of gaps in a sequence of sorted check numbers. An asterisk or similar character is typically attached to any check number that precedes or follows a gap, so that attention is immediately drawn to the missing check numbers.

The dialogue at right reveals a method for letting ENGLISH flag gaps in a check file (or any other file of supposedly sequentially numbered items). The first command in the dialogue simply shows some sorted check numbers. The next command uses a word named JUMPED that also shows the check numbers, but with an asterisk prefixed to any number that follows a "missing" number. In the example, 11 is flagged because 9 (and coincidentally 10) is missing, 8 is flagged because 7 is missing, 5 is flagged because 4 is missing, and 1 is flagged because there is no 0. The final command reveals how JUMPED is defined. Can you guess the effect of changing the minus sign to a plus in attribute 8 of JUMPED?

Of course, the method can be used on any type of file where knowing about missing numbers is important, and not just with item identifiers, but with any attribute.

Ⓟ

```
:SORT CHECKS
```

```
PAGE 1 18:17:10 16 JUL 1982
```

```
CHECKS....
```

```
1
2
3
5
6
8
11
12
```

```
8 ITEMS LISTED.
```

```
:SORT CHECKS JUMPED ID-SUPP
```

```
PAGE 1 18:17:21 16 JUL 1982
```

```
CHECKS....
```

```
*1
2
3
*5
6
*8
*11
12
```

```
8 ITEMS LISTED.
```

```
:CT DICT CHECKS JUMPED
```

```
JUMPED
```

```
001 S
002 0
003 CHECKS
004
005
006
007
008 A: IF (0-"1")(TCHECKS:X;:0)
    = "" THEN "*" : 0 ELSE 0
009 R
010 10
```

```
:
```

Ⓟ

wish list

Users of computer systems should always remember that the nice thing about software (besides the fact that it doesn't break when you drop it) is that programs are relatively easy to change — no soldering gun is necessary. So just because the operating system or the compiler or a system utility happens to work (or fail to work) in some particular fashion now, doesn't mean it has to always be that way. The next time an inspired idea arrives for improving your system, write it down and send it to Pragma for publication in the Wish List. Naturally, all such submittals are eligible under Pragma's author payment program.

1. Waking up a SLEEPing process. Besides sleeping until or for a specified time, it is also convenient to sleep until a certain "event." For example, it would be helpful to have the capability of allowing one process to sleep until another process finished executing certain commands or programs. Provide a WAKEUP verb to wakeup (in other words end SLEEP execution at) some other port if it is sleeping.

2. BREAK-ON values output with 'V' option. ENGLISH limits the length of break line values to 24 characters, which is too small. Allow break line values to be any length.

3. Object code and frame faults. A tempting but difficult task is to optimize BASIC programs by arranging the source code such that the resulting object code for loops and subroutines does not unnecessarily cross frame boundaries and cause excessive frame faults at execution time. Create a FRAME statement for BASIC programs that the programmer can insert in various places in the source code to cause the compiler to pad out the object code: the FRAME statement would tell the compiler that the object code for the following statements should start at the next frame boundary.

4. Error message cross-reference. It is often desirable to know which messages from ERRMSG can be generated once a TCL command executes from a proc, so that the proc may then test for errors with the IF E command. Unfortunately, there is no way to know all the error messages that may be generated. Provide a list of verbs showing the error messages each verb may generate.

5. Editor prestore command. It is often desirable to repeatedly execute the Editor's P command a certain number of times, but with only typing the command once. Provide a way to do so.

6. SCREENPRO display of multivalues. If a list of multivalues is given to SCREENPRO for display only, only the first can ever be seen. Allow control of which value in a list of multivalues will be displayed by SCREENPRO.

7. BASIC source inclusion. It is often desirable to have the same program segment (such as a list of equates defining symbolic attribute numbers) appear in many different programs. If one line (such as an equate) should change, then all programs must be re-edited, which is a slow and error-prone procedure. Allow a new statement (such as INCLUDE {file,} item) in BASIC programs so that the text in a specified item can be automatically included at that point in the source program. In this way, an item only needs to be edited once in

order to make the change in all necessary programs (in other words, programs with the INCLUDE statement).

8. Splitting lines with the Editor. It is often desirable to split an existing line into two lines with the Editor. Unfortunately, when this is attempted by placing an attribute mark in the middle of the line with the replace (R) command, the attribute mark truncates the line and the rest of the line (expected to be a new following line) is lost. Allow attribute marks to be inserted within a line, to cause a line to split into two lines.

9. Limiting logon attempts. A good security system puts limits on logon attempts, since remote programs can automatically try password combinations until successful. Record unsuccessful attempts to logon, and limit the number of attempts allowed.

10. Print limiting with ENGLISH. It is convenient to be able to suppress certain values from output by using commands such as

```
LIST PARTS QTY > "1000"
```

Unfortunately, print limiting does not work on attributes with F-conversions or on TOTAL lines. Make print limiting work in those situations.

11. BASIC character stream input/output. It is often desirable to write BASIC programs (such as word processors or cross-assemblers) that treat files as character streams. Unfortunately, BASIC input/output statements are record-oriented, which results in very slow and inefficient programs when they try to do stream-oriented input/output. The compiler and assembler appear to have a capability for fast character input/out, but it is virtually impossible to match their speed with a BASIC program. Provide a mechanism for fast character stream input/output in BASIC programs.

12. Using new operating system releases. It is often desirable to thoroughly test a new operating system release before converting over to it, but this is difficult because the machine can have only one operating system resident at a time. Allow more than one system version to be on disk at once, so that each port can select or automatically log on to a given release.

13. SORT-LABEL direction. SORT-LABEL is a convenient verb, but it is often desirable to have the order of the sorted items run vertically down each page instead of horizontally across each page. Give SORT-LABEL the option of printing or displaying with the order running vertically instead of horizontally.

14. **ELSE clause for ON statements.** Whenever an ON GOTO or ON GOSUB is used, it must usually be preceded by additional statements to verify that the control variable is in range. Expand the syntax of the ON statement to include an ELSE clause for when the control variable is out of range. For example:

```
10 INPUT COMMAND
   ON COMMAND GOTO 100,200,300 ELSE
     PRINT "INVALID COMMAND"
   GO TO 10
END
```

15. **BLOCK-CONVERT fonts.** The BLOCK-CONVERT file is designed to allow easy font changes. Unfortunately, the BLOCK-TERM and BLOCK-PRINT commands automatically refer to that file, so that having multiple font files is awkward. BLOCK-TERM and BLOCK-PRINT should allow specification of the file containing the block characters.

16. **File access and security.** If an account is given write access to a file, then the account can modify the base and modulo in the file's DL/ID pointer and bypass any system security measures. Allow dictionaries to be write protected while still allowing write access to the data level of the files.

17. **Sort by break results.** A file of inventory tags (the tag number is the item identifier) has a part number in attribute one, a location in attribute two, and a cost in attribute three. One typical report, easily done in ENGLISH, is:

PART	LOC	COST
1	A	10
1	B	20
1	C	30
TOTAL:		60
2	A	20
2	B	40
2	C	60
TOTAL:		120
3	A	15
3	B	10
3	C	15
TOTAL:		40

Unfortunately, ENGLISH cannot generate:

PART	TOT	DSND	COST
2		120	
1		60	
3		40	

Give ENGLISH the ability to generate this second report (in other words, sort on BREAK values).

18. **Changing Microdata baud rates.** To change a Microdata port's baud rate requires opening the machine, powering down, pulling out a card, and changing the setting on an unmarked DIP switch. Provide accessible switches or software to allow baud rates to be changed quickly and easily at any time.

19. **Preventing terminal logons.** It is often desirable to prevent all terminals except port zero from logging on. For example, after a system crash or file restore, it is convenient to be able to do certain housekeeping tasks before other ports log on. Provide a simple and quick method to prevent all ports other than port zero from logging on.

20. **Symbolic statement labels.** Numeric statement labels do not have any mnemonic value. Allow symbolic (non-numeric) statement labels. For example:

```
PARSE: GOSUB CHECK.INPUT
      IF ERROR THEN GO TO PARSE
```

21. **File access logging.** Have the system keep track of the last date and time that a file was read, written, or executed (one date and time for each type of access).

22. **POWER-FAILS verb.** The POWER-FAILS verb is a good way to find out that a power failure occurred, but without knowing when it occurred, it is of little use for tracking down periodic failures. Have the POWER-FAILS verb report the date and time of the last power failure.

23. **Logon data for ports.** It is often desirable to know the amount of activity that occurs on particular ports over time. Include the port number in the data that is saved in the ACC file at logoff.

□

Computing Modulos with ENGLISH

Dictionary words are presented for computing modulos with only ENGLISH conversions and correlatives.

Many installations use BASIC programs to compute file modulos, without realizing that ENGLISH is quite capable of doing the job on its own. The NEW.MOD word definition shown on the right is designed for the STAT-FILE dictionary. NEW.MOD will output the suggested modulo for a file described by a STAT-FILE item. A typical use of NEW.MOD would be in a command such as

```
LIST STAT-FILE ACCOUNT. FILE. MOD. NEW.MOD
```

to see the current and suggested modulos for each file. The DELTA word definition is a variant of NEW.MOD that outputs the difference between the current modulo and the modulo suggested by NEW.MOD, which is convenient for exception reports generated by a command such as

```
LIST STAT-FILE WITH DELTA # "0" MOD. NEW.MOD
```

which selects only those files with a modulo that needs adjusting.

Ⓟ

STAT-FILE Dictionary Words for Checking Modulos

```
NEW.MOD
001 S
002 4
003 Suggested
004
005
006
007 F;6;C500;7;R;C0;
    #;C500;7;/:;+;>;6
    ;*;6;C500;7;R;C0
    ;#;C500;7;/:;+;[;
    C500;7;R;C0;#;C5
    00;7;/:;+;*;+;"C
    2;_;R;C0;=;+;"C
    5;_;R;C0;=;+;"C
    2;_;R;C0;=;+
008
009 R
010 9
```

```
DELTA
001 S
002 99
003
004
005
006
007
008 F;4;6;C500;7;R;C
    0;#;C500;7;/:;+;>
    ;6;*;6;C500;7;R;
    C0;#;C500;7;/:;+;
    [;C500;7;R;C0;#;
    C500;7;/:;+;*;+;"
    ;C2;_;R;C0;=;+;"
    ;C5;_;R;C0;=;+;"
    ;C2;_;R;C0;=;+;-
009 R
010 9
```

Ⓟ

command files

Command files are typically the "glue" that holds the components of a software system together. Microdata's command file capability is called PROC, Prime offers Perform, and so on. In this regular department, Pragma will be presenting concepts and techniques to help installations squeeze the most out of their command file processors.

Building ENGLISH Headings with PROC

A sample console dialogue is presented, demonstrating the discovery of a typical programming bug involving the generation of ENGLISH headings.

Instead of wasting an output column on data that is constant for an entire report, a frequently used technique is to devise a proc that imbeds the constant data in the heading of the report being output, so that the data appears once in a heading at the top of each page and not in some column on the report. For example, a proc may input a part number and then output the part number, its description, and all of the part's purchase order receipt dates and quantities, but with the part number and description imbedded in the report heading, and with columns only being used for the receipt dates and quantities.

The dialogue shown on the next page is from an installation that uses just such a purchase order receipt proc. The system programmer has been informed that the PART.RECEIPTS proc, which has worked reliably for months, is suddenly crashing for certain input. The proc accesses two files, the RECEIPTS file and the PARTS file, which both use part numbers as item identifiers. The first attribute in items in the PARTS file is the description for each part.

The dialogue shows the programmer investigating the bug by first trying the proc on some typical input. After invoking the proc by typing "PART.RECEIPTS", the proc prompts back with "PART NUMBER?". The input "334" successfully generates a report, showing the heading "RECEIPTS FOR:", followed by the part number, its description (a card cage) and the report body itself: two columns of receipt dates and quantities. The proc then prompts for another part number. But when "541" is input, only error message [2] appears, there is no report, and the proc immediately prompts for another part number. Obviously something is bad about part 541.

The programmer hits just the RETURN key to get out of the proc, then uses the Editor to examine the proc's code. The programmer replaces the process (P) command in line 18 with the PP version to allow a check of exactly what kind of buffer output the proc is producing. Once again the proc is invoked and "541" is input to force the error, but now the PP command dumps the output buffer contents and prompts with a "?" to see if execution should continue.

Aha! Part number 541 happens to be a 10 inch cathode ray tube, and the inch mark (") in the part description, which the proc has duly included in the HEADING clause of the ENGLISH command being built, is causing an error message because the statement ends up with three double quote marks. Unfortunately, ENGLISH gets confused when HEADING text itself contains double quote marks, since quote marks are used as HEADING text delimiters.

What to do? First, the programmer answers the "?" prompt with "N" to abort the proc. As an emergency quick fix to the proc while still trying to retain the format of the proc's output so as not to confuse any users, the programmer uses the Editor to insert some additional code in the proc to chop off the description starting at any double quote mark, and then restores the P command in line 18. Invoking the proc one more time shows a very truncated description for part 541, but at least it doesn't crash, and part 334 still works fine, so now there's time for finding a good, long-term solution to the problem.

The technique of building variable HEADING text makes for nice reports, but this example shows what can happen when not enough thought is given to what might be encountered in the data being manipulated. Any proc that builds variable HEADING text must be sure to avoid double and single quotes, at least until ENGLISH is enhanced to allow more flexibility in the choice of HEADING text delimiters in a command.

[P]

:PART.RECEIPTS

PART NUMBER?334

RECEIPTS FOR:

334 CAGE, CARD, 2x4x4

DATE.... QTY...

08-06-82 2

11-04-82 5

PART NUMBER?541

[2] UNEVEN NUMBER OF SINGLE OR DOUBLE QUOTE-SIGNS (' ").

PART NUMBER?

:ED MD PART.RECEIPTS

TOP

.L99

001 PQN

002 100 RI

003 0

004 OPART NUMBER+

005 IP?

006 IF # A X

007 HLIST RECEIPTS =

008 A'

009 HHEADING "RECEIPTS FOR:'LL'

010 B

011 A\

012 FB (PARTS %1)

013 GO 200

014 MV %1 &1

015 A

016 200 H'L'"

017 H ID-SUPP

018 P

019 GO 100

EOF 19

.G18

018 P

.R

018 PP

.FI

'PART.RECEIPTS' FILED.

:PART.RECEIPTS

PART NUMBER?541

LIST RECEIPTS = '541' HEADING "RECEIPTS

FOR:'LL'541 CRT, 10", GREEN PHOSPHOR 'L'" ID-SUPP.

?N

:ED MD PART.RECEIPTS

TOP

.G14

014 MV %1 &1

.R/&1/&1,&1

014 MV %1 &1,&1

.I

014+IBH%1:G"1:

014+IF %1 = %2 F

014+

.G18

018 PP

.R

018 P

.FI

'PART.RECEIPTS' FILED.

:PART.RECEIPTS

PART NUMBER?541

RECEIPTS FOR:

541 CRT, 10

DATE.... QTY...

11-13-82 10

12-12-82 5

01-22-83 12

PART NUMBER?334

RECEIPTS FOR:

334 CAGE, CARD, 2x4x4

DATE.... QTY...

08-06-82 2

11-04-82 5

PART NUMBER?

:

queries

If you have questions or if you have answers, send either to Pragma for publication — both are eligible for Pragma's author payment awards.

1. We are on 3.2, and have been looking forward to upgrading to 4.1 to take advantage of the new binary save/restore facility. But now we are hearing reports that staying with 3.2 and the usual file saves may be the best way to go. Why?

Deciding between file saves and binary saves involves considering a variety of tradeoffs. If you have a black box disk controller (as opposed to a single board controller), then its no contest — the binary save software won't work. The binary save runs faster than a file save, but it can only execute in a stand-alone fashion with the system unavailable to other users. Selective restores are not possible from a binary tape, so doing a verify immediately after a binary save is important. Of course, a verify doubles the time spent reading tape, and if your save takes more than one reel, that means more time spent by the operator doing mounts and dismounts than without a verify, so that the time to do a binary save with verify begins to approach the time to do a file save. (File save tapes should be verified too, but many installations, especially those with multi-reel saves, don't always verify their file save tapes.) A binary save uses up more feet of tape, since the disk is saved on a cylinder basis, but it also gives a nice error summary that is helpful for keeping an eye on peripheral performance and degradation. Modulo-adjusting file restores cannot be done from binary saves.

It seems most installations have opted to continue using their old file save procedures, and to only use binary saves in emergencies and on special occasions.

2. The new Applications Language Liberator (ALL) and SCREENPRO both seem to be designed to do a lot of the same things — building screens, doing input processing, field validation, and so on. Are the two products in competition with one another? Is ALL so expensive because the rumor is true that it will be executing out of a special firmware board?

3. Why isn't anyone supposed to be on the system during file saves?

Users *can* be on during file saves, it's just that certain operations must be avoided. The rule you've heard is simply the easiest way to guarantee that no one breaks the *real* rules: leaving a group locked or simultaneously updating multiple related files. The file save processor locks groups before saving them on tape. If some update program is active at a port, say a sales order entry program, and it locks a group in the sales order file until the operator exits the program, then when the file save processor reaches the group the processor will hang and patiently wait for the group to unlock before it proceeds with the save. If the order entry operator has gone home for the day (having forgotten to exit the program and log off), it could be a long wait for the file save.

The second problem isn't so easy to recover from. Imagine two files: a vendor file containing a value indicating the amount due to be paid, and a cash receipt file indicating amounts that have been paid. If the two files happen to be situated on disk in such a way that considerable time elapses between the processing of each file by the file save operation, then the file save tape could end up with out-of-balance data if some other program is simultaneously being used to update the two files during the file save. For example, let's assume vendor #33 in the one file shows a balance of \$100 to be paid, and there is no entry in the receipt file for that amount. Now the file save begins, and it quickly finishes saving the complete vendor file to tape, but it will be awhile before the file save reaches the receipt file because there are so many other files on disk to save first (the order is determined by the order of file D-pointers in each master dictionary). Now a user logs on to enter a \$100 cash receipt for vendor #33. A program creates the receipt and changes the balance due for vendor #33 back to \$0 (remember this vendor's record has already been saved on tape). Now the file save finally reaches the receipt file and saves it on tape. But the results are out-of-balance. If a complete file restore were done off the tape, the user would discover the cash receipt has been created, but vendor #33 still shows a \$100 balance due. These kinds of multiple file updates must be avoided during file saves.

4. What is so great about the 4.1 spooler?

The best feature in the new spooler is that slave printers no longer need to be slaves, thereby allowing terminals to operate at higher baud rates. Before 4.1, a low-speed serial printer could only be included in a system as the spooler's console (one per system) or sharing a port with a terminal, which forced the terminal to run at a slow baud rate if output was to be sent continuously to the printer. With the new spooler, a serial printer can have its own dedicated port. A terminal running at 9600 baud can spool a report for the printer (typically a 300 or 1200 baud device), which will then print at the appropriate rate, while the terminal frees up to do more work at top speed.

A number of other features make the 4.1 spooler a much more powerful tool than the 3.2 version was, but there are still a few weak points, notably: the spooler doesn't seem to be able to keep its tables clean without an ABS load every week or so; the number of displayed frames for each print job increases as the job is built but doesn't decrease as it is printed; charges (lines printed, pages and forms used) are not collected; there is no verb for listing *only* the queue assignments or print jobs without also listing the action code menus (to allow custom menus that avoid options like EDIT PRINT JOB for security reasons). Undoubtedly these items will be improved in future releases. Overall, the 4.1 spooler is quite impressive.

□

Making Item Identifiers Hash Well

The effect on item hashing by ordering data within item identifiers is discussed, and the results of some empirical tests to demonstrate the effects are presented.

Probably the most overwhelming factor in the control of how "nicely" a given file of items will hash out is the selection of a good modulo for the file. But selection of a correct modulo is very much a last step in the tuning of a program and its data files. Is there some other criteria that system designers can take into account earlier in the implementation process that will help the system run optimally once implementation is complete?

Computer folklore claims that since the system's hashing algorithm accumulates item identifier characters from left to right, the rightmost item identifier characters tend to have more of an effect in the determination of the final hashed value. Therefore (recommends the folklore), database designers that specify the construction of item identifiers from concatenated values (such as concatenating an employee number and a date to form a unique identifier for each item in a file of employee time cards) should strive to make the right hand end of the item identifier vary over the widest possible range of values. That way the hashed results will tend to be as widely and evenly spread as possible, instead of clumped within small areas of the full modulo range.

For example, the item identifier in a file of vendor receipts could consist of a vendor number concatenated with the vendor's packing slip number for each receipt. If an asterisk is used to separate these two components, then there are two ways to specify and construct each identifier:

- (1) VENDOR * PACKSLIP
or
- (2) PACKSLIP * VENDOR

If only twenty different vendors are represented, then choice (1) would be more desirable. Since packing slip numbers will never be duplicated for one vendor, and probably rarely duplicated by other vendors, the rightmost digits of choice (1) will vary widely, while the rightmost part of choice (2) will always be a number from one to twenty. So according to theory, a file of item identifiers with technique (2) would, in general, hash more poorly than technique (1) — there would be more incidences of overflow and empty, unused frames.

How much of a difference does the order of data within item identifiers really make on file hashing? To find out, five tests were run which generated a certain number of item identifiers in two formats. First, a file was filled with "well designed" item identifiers. That is, each item identifier consisted of two pieces of data separated by an asterisk, and the wider ranging data was placed on the right end of each identifier. Then, the file was rehashed with the same identifiers, but with the wider ranging data placed in the left side of the identifiers to make a "poorly designed" set of file items. The format for the item identifiers in each test was:

Test 1: Receivers

Good identifier: VENDOR*PACKSLIP

Poor identifier: PACKSLIP*VENDOR

VENDOR is a number from 1 to 20, PACKSLIP is a string up to ten digits long.

Test 2: Cross-references

Good identifier: FILE*ATTRIBUTE

Poor identifier: ATTRIBUTE*FILE

FILE is a two letter file name, ATTRIBUTE is a name of from one to twenty letters.

Test 3: General ledger recap

Good identifier: MONTH*ACCOUNT

Poor identifier: ACCOUNT*MONTH

MONTH is a number from 1 to 12, ACCOUNT is a five digit number.

Test 4: Sparse matrix

Good identifier: ROW*COLUMN

Poor identifier: COLUMN*ROW

ROW is a number from 1 to 100, COLUMN is a number from 1 to 99,999.

Test 5: Transaction log

Good identifier: DATE*TIME

Poor identifier: TIME*DATE

DATE is the internal form of a date from 1/1/81 to 1/31/81 (4750 to 4780), TIME is the internal form of a time from 8:00 to 17:00 (28800 to 61200).

Care was taken to insure that the good and poor versions of each file totaled the same number of bytes and items. For all tests, a file of modulo 23 was used and 500 items, consisting of only the item identifiers themselves, were generated to fill the file. The following standard deviations were reported by the system for all tests:

Test	Item identifier	Std dev
1A	VENDOR*PACKSLIP	4.5
1B	PACKSLIP*VENDOR	4.7
2A	FILE*ATTRIBUTE	4.6
2B	ATTRIBUTE*FILE	4.6
3A	MONTH*ACCOUNT	3.4
3B	ACCOUNT*MONTH	4.7
4A	ROW*COLUMN	4.9
4B	COLUMN*ROW	4.1
5A	DATE*TIME	4.6
5B	TIME*DATE	5.5

The standard deviation is a measure of how evenly items are distributed throughout a file, with a lower value indicating the more desirable result. In four of the five test cases, the "good" identifier designs caused hashing as good as and usually better than the "poor" designs (which actually won in Test 4).

Although the differences in hashing caused by the arranging of data within an identifier might be considered insignificant, database designers should remember that it is easy to arbitrarily decide the order of identifier data in new file implementations, and tests such as those presented here indicate that a generally useful rule in optimizing file design is: arrange identifier data so that the rightmost characters will vary the most.



the computer room

A well-run computer room is the sign of an efficient data processing organization, and an important requirement in any computer room is good documentation — documentation regarding the care and use of peripherals, system error recovery, maintenance procedures, and a myriad of other details regarding day-to-day computer operations. In this regular department, *Pragma* will be presenting documentation, ideas, and techniques for guaranteeing a smooth-running computer room.

Mounting tapes quickly becomes second nature to the responsible individuals at a computer installation. Unfortunately, there is often only one person in an entire company who is familiar with tape-handling procedures. What if that person is suddenly unavailable? Are the tape procedures in use documented anywhere? Are tapes being handled in the correct fashion, to give them maximum lifetime?

Here is a set of steps describing how to mount and dismount tapes. The procedures is written for a Microdata single-density drive, but an installation can easily adapt it to their particular configuration.

MOUNTING A TAPE

The tape drive has five red status lights labeled WRITE ENABLE, LOAD POINT, READY, WRITE and READ. There are also five control buttons labeled POWER, LOAD, ON LINE, REWIND and RESET. The control buttons light up when turned on.

Your hands should be clean for this operation, to avoid harming the tape.

- 1 If a tape is currently mounted on the drive, find out who owns it and if it can be dismounted. Proceed to step 1 of the dismounting procedure if necessary. After dismounting, or if there is no tape already mounted, proceed to the following step.

- 2 Depress the POWER button to turn on the tape drive. The red READ status light should turn on.
- 3 Open the transparent front door of the tape drive and unlock the empty top hub of the drive by pulling out the flat black tension lever in the center of the hub.
- 4 Remove the reel you want to mount from any container or canister. Remove any plastic hanging seal from the edge of the magnetic tape reel, then reconnect the seal back into the shape of a circle to make it more manageable and to help prevent it from stretching flat and falling to the floor when set down.
- 5 Remove any adhesive tape or sponge which may have been used to prevent the magnetic tape from unwinding. Be sure that no adhesive is left on the magnetic tape surface.
- 6 **IMPORTANT:** If data is to be written on the tape you are mounting, insert a plastic write-ring in the back of the tape reel. If data is only going to be read from the tape, do not use a write ring.
- 7 Place the tape reel (write-ring side inward) on the empty top hub of the tape drive. Make sure the reel is as far back and as squarely on the hub as it can be. Only press the reel alongside its center hole. Do not press on the reel against the tape wound inside the reel. The red WRITE ENABLE status light will turn on if the write-ring is in the reel.
- 8 Lower the tension lever in the center of the hub to lock the reel in place.
- 9 Spin the upper reel in a clock-wise direction to unwind enough tape to wrap around the empty lower take-up reel. Do not let any tape touch the floor.
- 10 Wind the tape clock-wise around the lower take-up reel until you are manually able to rotate only the lower reel, and tape unwinds from the upper reel without slipping (3 or 4 full turns should be enough). Be sure to not fold or crease the tape on the lower reel.
- 11 Thread the tape through the tension arms and the read/write heads as shown in the diagram on the tape unit. Be sure the tape lies correctly within all the guides. Touch the tape as little as possible. Do not touch the inside surface of the tape.
- 12 Depress the LOAD button. Tension will be applied to the tension arms.

- 13 Depress the LOAD button a second time. The tape reels should slowly wind in a clock-wise direction and stop within 10 seconds. If so, go to the following step. If not, depress the REWIND button. The tape reel should wind in a counter-clockwise direction at high speed and then stop.
- 14 The red LOAD POINT and red READ status lights should be on. The READY and WRITE lights should be off. If a write-ring is present, the red WRITE ENABLE status light should be on. Depress the ON LINE button. The indicator lamp in the ON LINE button and the red READY status light should now be on. Shut the tape drive door. The tape is ready.

DISMOUNTING A TAPE

The tape drive has five red status lights labeled WRITE ENABLE, LOAD POINT, READY, WRITE and READ. There are also five control buttons labeled POWER, LOAD, ON LINE, REWIND and RESET. The control buttons light up when turned on.

Your hands should be clean for this operation, to avoid harming the tape.

- 1 Observe the tape drive unit. The following steps assume a tape is currently mounted with the POWER control button lit.
- 2 If the ON LINE button is lit, depress it to make the ON LINE button and the red READY status light go off. The ON LINE button alternates between on and off with each depression.
- 3 Depress the REWIND button. The REWIND button may light and the tape reel will spin in a counter-clockwise direction. The tape will then stop and the REWIND button should go off if lit.
- 4 If the red LOAD POINT status light is on, go back to step 3.
- 5 Open the transparent front door of the tape drive unit and manually turn the upper tape reel to remove the tape from the tension arms and read/write heads.
- 6 Depress the POWER button to turn off the tape drive.
- 7 Lift the flat black tension lever in the center of the top hub holding the tape reel and remove the reel. Note: if a dismounted tape feels very warm, it is a sign of high temperature within the mainframe cabinet. Check that the cabinet's air filter is not clogged with dust.
- 8 Lower the tension lever in the center of the top hub and close the tape drive's door.
- 9 Replace any tape or sponge for keeping the tape from rewinding. Tape is not recommended because it can leave adhesive on the magnetic tape surface. Reseat any plastic seal around the edge of the tape reel. Be sure the seal is labeled the same as the reel. Remove any write-ring and return the tape to its storage location.

■

IN THE NEXT ISSUE: trouble trees

letters

If you use a Pick system, **Pragma** wants to hear from you. Have you developed applications of interest to other users? Do you plan to acquire new hardware? What features would you like to see in **Pragma**? Are you active in any type of user group organization?

Write **Pragma** today. All letters to the Editors are welcome, and as many as possible will be published in the Letters Department in every issue.

■

Patching the Exit Problem in 3.2 SCREENPRO

A procedure for patching 3.2 SCREENPRO is described. The patch corrects a bug that causes the ELSE clause of INPUT USING statements to be unconditionally executed.

The 3.2 release of Microdata's system software included a major new addition: SCREENPRO. Like other first versions of ambitious software, SCREENPRO unfortunately included a number of critical bugs. The arrival of 3.2B brought enough improvements in SCREENPRO so that it could be reliably used as long as the remaining known bugs were carefully avoided by the programmer. One such bug could be particularly devastating, in that it would cause the ELSE clause of any INPUT USING statement to *always* be taken. Since the ELSE clause usually controls the exit path out of a screen and its calling program, the bug tends to make a program prematurely loop or quit, leaving no way for the operator to continue on to subsequent prompts.

What is particularly interesting about the bug is that it tends to suddenly appear after a visit by a customer engineer who has performed diagnostic tests! That is because the bug is caused by zeroes being present in certain areas of memory reserved for each port. If these areas are non-zero, SCREENPRO INPUT USING statements work correctly. But if these certain areas in memory are somehow cleared to zeroes, INPUT USING statements suddenly begin to always execute their ELSE clauses. Warmstarts, coldstarts and ABS loads will have no effect — the memory locations will have to be manually patched. Once patched, the only way memory has typically ever been observed to revert back to zeroes has been because a customer engineer visited and ran certain diagnostics via the console switches, thereby clearing or garbaging memory in such a way as to prevent SCREENPRO from properly initializing.

The best way to avoid 3.2 SCREENPRO bugs is to upgrade to 4.1, but a number of installations remain on 3.2 for a variety of reasons. If such installations are finding it difficult to effectively use 3.2 SCREENPRO because of the ELSE exit bug, they should try the patch described here:

- 1 Log on to an account that has the WHERE verb and SYS2 privileges, such as the SYSPROG account.

- 2 Type the command

WHERE (A)

and hit the RETURN key to get a list of primary control block (PCB) numbers. For example, the list might look like

PORT	PS	RTN	STACK..
00 0200	7D	6.086	6.03C 5.064
01 0220	7D	6.086	6.03C 5.064
02 0240	7F	107.114	107.0D9
03 0260	7D	6.086	6.03C 5.064
04 0280	7D	6.086	6.03C 5.064
05 02A0	7D	6.086	6.03C 5.064
06 02C0	7D	6.086	6.03C 5.064

07 02E0	7D	6.086	6.03C	5.064
*08 0300	7F	121.000	121.07D	
09 0320	7D	4.001	6.03C	43.07E
10 0340	7D	6.086	6.03C	5.064
11 0360	7D	6.086	6.03C	5.064
12 0380	7D	6.086	6.03C	5.064
13 03A0	7D	6.086	6.03C	5.064
14 03C0	7D	6.086	6.03C	5.064
15 03E0	7D	6.086	6.03C	5.064

The patch must be installed for each port that will use SCREENPRO. Assume the patch needs to be installed for port 13. In this list, 3A0 is the hexadecimal PCB number for port 13.

- 3 Use the ADDX command to add 1D hexadecimal to the PCB number. For example, typing

ADDX 3A0 1D

and hitting the RETURN key will cause

3BD

to be displayed.

- 4 Enter the patch address by first hitting the BREAK key to get the debugger's exclamation point prompt. Then type

- a period followed by
- the result displayed by the ADDX command followed by
- .17E;6 followed by
- hitting the RETURN key

to get a display of memory. For example, entering

!.3BD.17E;6

(the ! came from hitting the BREAK key) and hitting the RETURN key will then cause a period, twelve characters (usually but not always zeroes) and an equal sign to be displayed. For example,

!.3BD.17E;6.000000000000 =
└─RETURN

is a typical display.

- 5 Now type

.FFFFFFFFFFFF

(a period and twelve F's) and hit the RETURN key. For example,

!.3BD.17E;6.000000000000 = .FFFFFFFFFFFF
! └─RETURN RETURN└─

- 6 The debugger will continue to prompt with exclamation points. You can exit the debugger by typing END and hitting the RETURN key.

- 7 These steps show how to install one patch for one port using its PCB number from the WHERE list as a starting point. The patch must be repeated for each port that will be using SCREENPRO (using the appropriate PCB number for the port from the WHERE list).

P



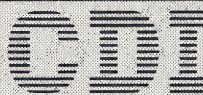
MORE POWER TO YOU.

No one can argue with the raw power available in the IBM Series/1 minicomputer. Or that its performance and reliability make it highly cost effective.

To realize the full potential of the Series/1, all it required was an equally powerful operating system. Like PICK.

Under various trade names, PICK applications are widely known and

CDI
unleashes
the **IBM**
Series/1.



COMPUTER DISTRIBUTORS, INC.

appreciated. On Microdata, for example, PICK is REALITY. With its simple-sentence structure, it is the most user friendly data base software in existence.

Now, in an unprecedented union, CDI brings these two industry front runners together.

The best. And the brightest. Never before has so much power been so concentrated or easier to use.

Join these exclusive dealers.

CDI is looking for highly qualified dealers to market the PICK implemented IBM Series/1.

For inquiries, call CDI toll-free at (800) 426-8931.

Or write Computer Distributors, Inc., 1309 114th Avenue S.E., Suite 300, Bellevue, WA 98004.

DATA ENTERPRISES OF THE NORTHWEST, Seattle, WA, (206) 762-9474 • DIVERSIFIED COMPUTER CORP., Seattle, WA, (206) 223-9326 • COFFMAN SYSTEMS, Burlingame, CA, (415) 692-1022 • MRP, INC., Santa Ana, CA, (714) 972-2000 • TRANSITION SYSTEMS, Phoenix, AZ, (602) 829-2145 • COMPUTER DATA-RESOURCES, Denver, CO, (303) 388-4041 • IMPROVED INSURANCE SYSTEMS, INC., Chicago, IL, (312) 782-6520 • SYSTEMS MANAGEMENT, INC., Rosemont, IL, (312) 298-3840 • RAO ASSOCIATES, INC., Birmingham, MI, (313) 540-2211 • EDP OF AMERICA, INC., Cranford, NJ, (201) 272-0770
Distributor: PARAGON COMPUTER GROUP, LTD., Coquitlam, B.C., Canada, (604) 939-6461.

games

For many people, computers are synonymous with games, now that the video game industry has become such a giant. Programmers frequently cut their teeth on small game programs, since such programs are often straight-forward and self-contained without a lot of complicated interfaces to files or other software. And, more than anything, games are simply entertaining and fun.

The Games Department will be making periodic appearances in Pragma. If you have a game program you would like to share, send it in for publication. If there's anything the Pragma staff has time to do, it's "performing rigorous software quality assurance" (in other words, trying out a new game on the computer).

The Shell Game

Yes, ladies and gentlemen, the CRT is quicker than the eye! Step right up and try your hand with Lady Luck! Try to keep your eye on the flashing little asterisk, and watch your CPU cycles melt away like magic! Step right up, ladies and gentlemen, step right up!

In the classic shell game, a pea is hidden under one of three walnut shell halves, which are then shuffled. The object of the game is to correctly guess which shell hides the pea after shuffling.

In this computer version, an asterisk appears on the screen in place of a pea, and three "boxes" are used instead of walnut shells. In real life, the game suffers notoriously from sleight-of-hand tricks. In this version, the program is completely honest. If you can keep your eye on the box containing the asterisk, you can win every time.

The program is named SHELL and is written in Microdata DATA/BASIC (see listing on page 39). The variable PASSES (initialized in line 1 of the program) tracks the number of times the game has been played, while RIGHT (initialized in line 2) stores the number of times the player correctly guesses where the asterisk is, so that the program can report the player's percentage of correct guesses at line 114.

SHUFFLES controls the number of times two boxes are swapped before the player is asked to guess the asterisk location. Initially (line 3) SHUFFLES is five, but line 121 bumps SHUFFLES after each play, so the second game is six shuffles, the next game is seven shuffles, and so on for as long as play continues, requiring the player to track the asterisk longer and longer with each play.

The action for each game starts with the clearing of the screen at line 5. The three variables BOXA, BOXB, and BOXC (initialized in line 6) track the actual asterisk location. If the asterisk is in a given box, that variable will be set to 1 ("true") while the other boxes will be 0 ("false"). For example, if the asterisk is in the middle box, we have BOXA=0, BOXB=1 and BOXC=0.

To handle the screen animation, the program divides the display into nine regions:

1	2	3
4	5	6
7	8	9

The three boxes begin at locations 4, 5 and 6. Lines 7 through 10 of the program "paint" the boxes at the appropriate starting spots on the screen by calling subroutine 200 which starts at line 124. The subroutine accepts a box position number (set from 1 to 9 in variable POS) and a character to paint (stored in LTR) and then either erases or paints the box at the appropriate position on the screen. If LTR is blank, a box is effectively erased. If LTR is non-blank (the character "X" is used in the program), a 2 by 3 character box is output and appears on the screen. The routine executes a do-nothing FOR-loop 30 times after painting or erasing a box to keep the action moving slow enough to see. (Thirty iterations were chosen for a Microdata 6000 with no other ports active.)

The program chooses which box the asterisk starts under in line 11, where INIT is randomly set to 1, 2 or 3. The appropriate box is momentarily lifted by lines 12 to 34 to reveal the asterisk at its starting position. For example, if the asterisk starts in the first box, then INIT is 1 and lines 14 through 19 are executed: BOXA is set true (line 14) to remember where the asterisk is, the box is erased from POSITION 4 (line 15), the asterisk is shown (line 16), the box is displayed at POSITION 1 (line 17), erased (line 18), and put back at POSITION 4 (line 19) to cover up the asterisk. Similar animation occurs when either of the other two cases starts the game.

Now things really start moving. The FOR-loop from line 35 to line 103 does the shuffling. A shuffle is the swapping of positions of any two boxes. There are three boxes, so there are six possible different swamps, one of which is chosen by setting SWAP in line 36. All six swamps execute similarly by calling subroutine 200 to alternately erase and paint boxes at various positions on the screen in order to make the boxes appear to move. The TEMP variable is used to move the truth value be-

SHELL PROGRAM LISTING

```

001  PASSES = 0 ;* Number of times Played
002  RIGHT = 0 ;* Number of times Player guessed right
003  SHUFFLES = 5 ;* Number of times boxes should be shuffled
004  100 * Start a game
005  PRINT CHAR(12): ;* Clear screen
006  BOXA = 0 ; BOXB = 0 ; BOXC = 0 ;* Set true when box has *
007  LTR = "X" ;* Letter to paint
008  POS = 4 ; GOSUB 200 ;* Paint boxes
009  POS = 5 ; GOSUB 200
010  POS = 6 ; GOSUB 200
011  INIT = RND(3)+1 ;* Choose box
012  BEGIN CASE ;* Lift box to reveal *
013  CASE INIT = 1
014      BOXA = 1
015      LTR = " " ; POS = 4 ; GOSUB 200 ;* Remove box
016      PRINT @(5,11):"*": ;* Show *
017      LTR = "X" ; POS = 1 ; GOSUB 200 ;* Show box lifted
018      LTR = " " ; GOSUB 200 ;* Drop box
019      LTR = "X" ; POS = 4 ; GOSUB 200 ;* Back as was
020  CASE INIT = 2
021      BOXB = 1
022      LTR = " " ; POS = 5 ; GOSUB 200 ;* Remove box
023      PRINT @(11,11):"*": ;* Show *
024      LTR = "X" ; POS = 2 ; GOSUB 200 ;* Show box lifted
025      LTR = " " ; GOSUB 200 ;* Drop box
026      LTR = "X" ; POS = 5 ; GOSUB 200 ;* Back as was
027  CASE 1 ;* (INIT=3)
028      BOXC = 1
029      LTR = " " ; POS = 6 ; GOSUB 200 ;* Remove box
030      PRINT @(17,11):"*": ;* Show *
031      LTR = "X" ; POS = 3 ; GOSUB 200 ;* Show box lifted
032      LTR = " " ; GOSUB 200 ;* Drop box
033      LTR = "X" ; POS = 6 ; GOSUB 200 ;* Back as was
034  END CASE
035  FOR I = 1 TO SHUFFLES ;* Swap boxes
036      SWAP = RND(6)+1
037      BEGIN CASE
038      CASE SWAP = 1 ;* A to B
039          TEMP = BOXB
040          BOXB = BOXA
041          BOXA = TEMP
042          LTR = " " ; POS = 4 ; GOSUB 200
043          LTR = "X" ; POS = 2 ; GOSUB 200
044          LTR = " " ; POS = 5 ; GOSUB 200
045          LTR = "X" ; POS = 4 ; GOSUB 200
046          LTR = " " ; POS = 2 ; GOSUB 200
047          LTR = "X" ; POS = 5 ; GOSUB 200
048      CASE SWAP = 2 ;* A to C
049          TEMP = BOXC
050          BOXC = BOXA
051          BOXA = TEMP
052          LTR = " " ; POS = 4 ; GOSUB 200
053          LTR = "X" ; POS = 2 ; GOSUB 200
054          LTR = " " ; POS = 6 ; GOSUB 200

```



```

055      LTR = "X" ; POS = 8 ; GOSUB 200
056      LTR = " " ; POS = 2 ; GOSUB 200
057      LTR = "X" ; POS = 6 ; GOSUB 200
058      LTR = " " ; POS = 8 ; GOSUB 200
059      LTR = "X" ; POS = 4 ; GOSUB 200
060      CASE SWAP = 3 ; * B to A
061          TEMP = BOXA
062          BOXA = BOXB
063          BOXB = TEMP
064      LTR = " " ; POS = 5 ; GOSUB 200
065      LTR = "X" ; POS = 1 ; GOSUB 200
066      LTR = " " ; POS = 4 ; GOSUB 200
067      LTR = "X" ; POS = 5 ; GOSUB 200
068      LTR = " " ; POS = 1 ; GOSUB 200
069      LTR = "X" ; POS = 4 ; GOSUB 200
070      CASE SWAP = 4 ; * B to C
071          TEMP = BOXC
072          BOXC = BOXB
073          BOXB = TEMP
074      LTR = " " ; POS = 5 ; GOSUB 200
075      LTR = "X" ; POS = 3 ; GOSUB 200
076      LTR = " " ; POS = 6 ; GOSUB 200
077      LTR = "X" ; POS = 5 ; GOSUB 200
078      LTR = " " ; POS = 3 ; GOSUB 200
079      LTR = "X" ; POS = 6 ; GOSUB 200
080      CASE SWAP = 5 ; * C to A
081          TEMP = BOXA
082          BOXA = BOXC
083          BOXC = TEMP
084      LTR = " " ; POS = 6 ; GOSUB 200
085      LTR = "X" ; POS = 2 ; GOSUB 200
086      LTR = " " ; POS = 4 ; GOSUB 200
087      LTR = "X" ; POS = 8 ; GOSUB 200
088      LTR = " " ; POS = 2 ; GOSUB 200
089      LTR = "X" ; POS = 4 ; GOSUB 200
090      LTR = " " ; POS = 8 ; GOSUB 200
091      LTR = "X" ; POS = 6 ; GOSUB 200
092      CASE SWAP = 6 ; * C to B
093          TEMP = BOXB
094          BOXB = BOXC
095          BOXC = TEMP
096      LTR = " " ; POS = 6 ; GOSUB 200
097      LTR = "X" ; POS = 2 ; GOSUB 200
098      LTR = " " ; POS = 5 ; GOSUB 200
099      LTR = "X" ; POS = 6 ; GOSUB 200
100      LTR = " " ; POS = 2 ; GOSUB 200
101      LTR = "X" ; POS = 5 ; GOSUB 200

```

SHELL LISTING CONTINUED

```

102      END CASE
103      NEXT I
104      PASSES = PASSES+1
105      PRINT
106      PRINT
107      PRINT "Which box has the *? Box 1, 2 or 3":
108      INPUT ANSWER
109      IF (BOXA AND (ANSWER=1)) OR (BOXB AND (ANSWER=2)) OR (BOXC
AND (ANSWER=3)) THEN

```


SHELL LISTING CONTINUED

```

110 RIGHT = RIGHT+1
111 PRINT "Correct!"
112 END ELSE PRINT "Wrong."
113 PRINT
114 PRINT "You have been correct ":
115 PRINT INT(RIGHT/PASSES*100):
116 PRINT "% of the time."
117 PRINT
118 PRINT "Quit? (Y or N)":
119 INPUT ANSWER
120 IF ANSWER = "Y" THEN STOP
121 SHUFFLES = SHUFFLES+1
122 GO TO 100
123 *
124 200 * Fill or clear box
125 X = MOD((POS-1),3)*6+4
126 Y = INT((POS-1)/3)*4+6
127 PRINT @(X,Y):STR(LTR,3):
128 PRINT @(X,Y+1):STR(LTR,3):
129 FOR J = 1 TO 30 :* Delay
130 NEXT J
131 RETURN
132 *
133 END

```

®

tween BOXA, BOXB and BOXC, indicating the actual position of the asterisk so the program can remember it and compare with the player's guess.

For example, if SWAP turns up equal to 6, then line 92 is reached. Lines 93 to 95 make sure the state of BOXB and BOXC are correctly exchanged in case one of them has the asterisk. Then lines 96 to 101 do the animation: the right box moves up over the middle (from POS 6 to POS 2), the middle box jumps right (POS 5 to POS 6), and the first box then falls into the middle spot (POS 2 to POS 5).

After all the shuffles finish, PASSES is bumped in line 104 and the player is prompted for a guess. If the guess is correct, then line 109 tests true and RIGHT is bumped in line 110. The player's percentage is then reported, and the player is prompted for permission for another game. Before another game starts, SHUFFLES is bumped in line 121 to try and make the next round a little more difficult.

Be sure to add the SHELL game to your system's collection of games, and don't forget to fine tune SHELL for your system's throughput by adjusting the delay in line 129. SHELL looks best on terminals running at 9600 baud.

® IN THE NEXT ISSUE: fortune cookies

SEMAPHORE CORPORATION

Provides Software and Support for Microdata Systems

- COMPLETE SPECIFICATION
- ELEGANT DESIGN
- SYSTEMATIC IMPLEMENTATION
- THOROUGH TESTING
- PATIENT USER TRAINING
- ONLINE DOCUMENTATION
- PROVEN POLICIES AND PROCEDURES

SEMAPHORE

SEMAPHORE CORPORATION
207 GRANADA DRIVE
APTOS, CA 95003
408-688-9200

SPECIALIZING IN MANUFACTURING SYSTEMS: ENGINEERING • PURCHASING
• RECEIVING • INSPECTION • ORDER ENTRY • SHIPPING • CUSTOMER SERVICE
• PRODUCTION CONTROL • SHOP FLOOR • PAYROLL • PERSONNEL
• RECEIVABLES • PAYABLES • COST ACCOUNTING • GENERAL LEDGER
• FIXED ASSETS • MODELING

The Simple, Successful Solution!

Upgrade Your Microdata Equipment To Evolution's High Standards

Over 75 Successful Upgrades To Date! *

180/01 — Microdata 1600 to Evolution R-80
180/02 — Microdata Royale to Evolution R-80
180/03 — Microdata to Evolution CPU
180/07 — Evolution R-77 to R-80
180/08 — Evolution R-77 to R-80E
Extended Memory Model

Upgrade Features:

- Operating System Upgrade speeds Terminal Response Time up to 20%
- Extended Memory Operation speeds Terminal Response Time up to 200%
- Up to 180% faster Program Run Times
- Supports up to 64 serial ports for greater on-line terminal capacity
- Improves system throughput by allowing extension of real memory beyond 64KB up to 1024KB
- Supports Error Correcting MOS memory in 256KB increments for higher performance at lower cost
- Expanded spooler system supports up to 4 parallel and 16 serial printers providing unmatched configuration flexibility
- Supports larger files through disk storage systems providing up to 1.6 Billion bytes of on-line data
- DMA Tape Drive conversion provides tape I/O without CPU overhead
- Parallel printer controllers will drive line-printers at maximum capacity, accommodating any high volume printing requirement
- R-80 BASIC compiler automatically converts software to expanded system while reducing program run-time
- Improved data base Query Processor for faster on-line response
- Extended Query Processor language improves data base inquiry functions
- Simplified and extended operation of the Runoff text-formatter
- Improved assembly and BASIC debuggers simplify new software development
- New TCL level interface simplifies integration of non-standard terminals
- Enhanced security systems
- Nationwide field service

** Upgrade Reference Sites Available — Some Features May Require an Evolution CPU*

Hardware, Firmware, and Software products to enhance your present Evolution or Microdata System.

EVOLUTIONTM
COMPUTER SYSTEMS CORPORATION

For Further Information Contact Ed Towers or Chris Millington

250 East Emerson Avenue, Orange, California 92665, U.S.A. ■ (714) 974-7670 ■ TWX 910-593-1613

NEW

A software package that gives you instant, accurate, up-to-date accounting information from your mini computer.

GAAP

FINANCIAL SOFTWARE TM

IF YOU HAVE

ADDS Mentor Evolution
Honeywell Ultimate Microdata Prime
or any **PICK**-operating system,

this is the best* financial package you can buy. GAAPTM has a full warranty, complete documentation, password security control, and free upgrades for 1 year.

GENERAL LEDGER

Wait no more for trial balances. With GAAPTM, you can have financial reports every day of the year. Year-to-date, period-to-date balance and budget analysis (on a cash or accrual basis) are a simple selection from the software menu.

ACCOUNTS PAYABLE

Provides Vendor File Analysis, Recurring and Accrued Payables, Cash Requirements Schedule, Manual Disbursements and Automatic Check Issuance, Automatic Bank Reconciliation, and much, much more.

ACCOUNTS RECEIVABLE

Produces ready-to-mail statements on a balance forward or open invoice basis. On-demand reports include Aged Trial Balance, Open Invoice Listings, Cash Receipts and Adjustments Journal, and Delinquent Account Analysis.

PAYROLL INTERFACE

As part of the General Ledger system, GAAPTM provides automatic posting of payroll information from magnetic tape for those using service bureau payroll systems.

OTHER OPTIONS

- Purchasing/Receiving Interface
- Inventory Control Interface
- Program and Report Generator (DEFINE II)TM
- Custom Software

Turn your accounting system into a management tool, every day of the year! Call us today for more information.



KDK Enterprises, Inc.
Suite 302
1491 Chain Bridge Road
McLean, VA 22101
703/893-7883

*A Washington, D.C. auditing firm found that GAAPTM, as the name implies, truly follows Generally Accepted Accounting Principles.

DO YOU ADVERTISE?
GAIN EXPOSURE TO A LARGE RESPONSIVE AUDIENCE BY ADVERTISING IN PRAGMA

FOR RATES AND LAYOUT INFORMATION WRITE:

ADVERTISING MANAGER
PRAGMA
207 GRANADA DRIVE
APTOS, CA 95003
OR TELEPHONE 408-688-9200

The Key



to greater utilization of your Microdata Computer

For more information contact:

MICRU International

4300-L Lincoln Avenue
Rolling Meadows, IL 60008

Shebesta

Associates Inc.

7715 Beechmont Avenue, Suite 4, Cincinnati, Ohio, 45230

USED MICRODATA EQUIPMENT

WE HAVE IN STOCK:

* 50 MB REFLEX DRIVES	\$ 9000.00	* PRINTER CONTROLLERS	\$1595.00
* 10 MB DRIVES	\$ 2250.00	* PRINTERS	
* 16 K MEMORY	\$ 900.00	* PRISMS	\$ 998.00
* 8 WAY BOARDS	\$1595.00	* COMPLETE SYSTEMS	

WE ARE SPECIALISTS IN THE PICK OPERATING SYSTEM

NEW **PRINTRONIX PRINTERS**

150 LPM \$4985.00

300LPM \$5975.00

600 LPM \$7780.00

DATA PRODUCTS PRINTERS

MICRODATA
PRIME
HONEYWELL

300 LPM \$6900.00

600 LPM \$7850.00

ADDS

Applied Digital Data Systems Inc.

VIEWPOINT

(WITH THE PURCHASE OF A NEW PRINTRONIX OR DATA PRODUCTS PRINTER)

FREE
DISPLAY TERMINAL

CALL
CHUCK HAAS

800-247-2000 Ext. 189
IN OHIO 513 - 232 - 5000

\$25

WILL BUY YOU:

- 2800 sheets of low-grade 14x11 printer paper
OR
- 1 copy of the Microdata Programmer's Reference Manual
OR
- 2400 feet of magnetic tape
OR
- 30 minutes of an inexpensive computer consultant's time
OR

• **YOUR FIRST COPY OF PRAGMA!** Pages and pages packed with useful, innovative, pragmatic articles and software: general purpose tools, algorithms, problem-solving techniques, utilities, applications—even the ads are informative.

You can't find a better value,
SO SUBSCRIBE TODAY.

☐ 1 YEAR (4 issues) \$100⁰⁰

☐ 1 YEAR (Foreign) \$152⁰⁰
U.S. FUNDS ONLY

☐ YES, you may publish and announce my name as a new subscriber in the next issue.

My subscriber number on this issue's address label is _____.

Name _____
PLEASE PRINT

Company _____

Address _____

City _____ State/Province _____

Country _____ ZIP/Mail Code _____

Telephone _____

I was referred by
paid subscriber
number _____

Please extend their
subscription by one
FREE issue.

Enclose payment and
mail to:

PRAGMA
207 Granada Drive
Aptos, CA 95003



INTERACTIVE SYSTEMS PRESENTS

3 SOFTWARE PRODUCTS THAT WILL SAVE YOU TIME AND MONEY:

1. TCL-AUDITOR.....\$ 149

TCL-AUDITOR is a TCL pre-processor which provides each terminal user with a list of the previous 64 sentences entered at TCL. Any of these previous sentences may easily be listed, corrected, modified, expanded, re-executed, or saved without re-entering the sentence. **TCL-AUDITOR** saves the terminal user from the tedious process of repetitive keying of incorrectly entered sentences.

TCL-AUDITOR will not only save hours of frustration due to re-keying, it will provide an audit trail of the activity on each port. When you must know "what was really entered" in order to correct some problem situation, **TCL-AUDITOR** will provide the information you need.

2. REPORT-GEN.....\$ 395

How many times have you had a situation where you wanted to create a report and **ENGLISH** could handle everything but one relatively small part? Therefore, because of the one small requirement, you had to write a complete print program. Now with **REPORT-GEN**, you can generate a BASIC program from an **ENGLISH** sentence that comes closest to your needs, and then modify the program to perform that one small task that **ENGLISH** could not handle.

REPORT-GEN will save you many hours of programming time. It operates upon the standard dictionary items used by **ENGLISH**. There is no modification of your dictionary or data file required. **REPORT-GEN** prompts the operator for the program name, program file, and the sentence to be generated into BASIC code. The generated code is well documented and easy to modify.

3. SCREEN-GEN\$1195

SCREEN-GEN is an inexpensive yet comprehensive software tool used for designing, documenting, and generating screens and screen programs.

With **SCREEN-GEN** you can dynamically design and construct screen images, display fields, prompts, and validation criteria for a data entry process. Then **SCREEN-GEN** will generate a BASIC program that uses the defined screens in updating a data file. **SCREEN-GEN** can produce multi-screen programs, provide for multi-valued data entry and file attribute structures (in both associated and unassociated formats), generate full documentation on each defined screen, and generate efficient top-down structured code that can be easily modified.

SCREEN-GEN is easy to learn and will decrease programming and maintenance time drastically.

**For information or to order,
call (602) 993-3579 today!**



INTERACTIVE SYSTEMS
129 EAST VOLTAIRE AVE.
PHOENIX, ARIZONA 85022

IN THE NEXT **PRAGMA**

SYSTEM SECURITY • VANILLA:
BILLS OF MATERIAL • BLACK
BOX FORMATING • SYSMAP:
FILE FORMATS • A CUSTOM
PIECE OF HARDWARE FOR THE
COMPUTER ROOM • AN
INTRODUCTION TO ENGLISH,
PART 1 • AUTOMATIC MODULO
SETTING • LETTERS, WISH
LISTS, BUG REPORTS
AND MORE...

SUBSCRIBE TO PRAGMA NOW

- ☐ 1 YEAR (4 issues) \$100⁰⁰
☐ 1 YEAR (Foreign) \$152⁰⁰
U.S. FUNDS ONLY

☐ YES, you may publish and announce my name as a new subscriber in the next issue.

My subscriber number on this issue's address label is _____.

Name _____
PLEASE PRINT
 Company _____
 Address _____

 City _____ State/Province _____
 Country _____ ZIP/Mail Code _____
 Telephone _____

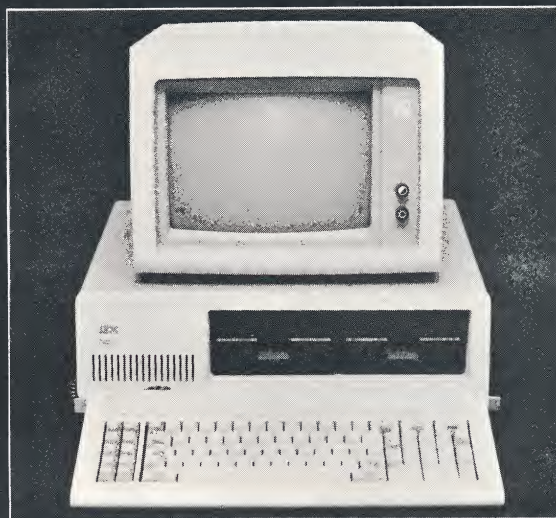
I was referred by
paid subscriber
number _____.

Please extend their
subscription by one
FREE issue.

Enclose payment and
mail to:

PRAGMA
207 Granada Drive
Aptos, CA 95003

IBM's Personal Computer Talks to Your Minicomputer Via **R/NET**



You can connect an IBM PC equipped with R/NET to your Reality, Ultimate, Mentor, Evolution, or Information relational data base machine. R/NET executes your existing software without modification. In addition to supporting cursor control for your interactive programs, you can either print the current screen contents or engage the printer for printing a multi-page report. And under operator or program control, R/NET will capture data sent to the IBM PC on its own disk storage.

Now you can let VisiCalc do your financial modeling. Execute EasyWriter on your dedicated microcomputer for your wordprocessing requirements. Your PC is multi-lingual, i.e., BASIC, Cobol, Pascal, Forth, FORTRAN, C, and the 8086 Macro Assembler. Application packages like General Ledger, Accounts Receivable, Accounts Payable, Inventory Control, Job Costing, and Payroll are available today.

R/NET is distributed on its own diskette and comes complete with documentation manual and interface cable for \$200. A typical PC configuration retails for \$3175. Make your next terminal an IBM Personal Computer, the most respected name in the industry.

cosmos

Direct your orders to: Cosmos Inc.,

10626 148th Avenue S.E., Renton, WA 98056 • (206) 226-9362

Think Relational

• VisiCalc TM of VisiCorp • EasyWriter TM of Information Unlimited Software Inc. • R/NET TM of Cosmos Inc. • IBM TM of International Business Machines • Ultimate TM of Ultimate Corp. • Mentor TM of Applied Digital Data Systems • Evolution TM of Evolution Corp. • Information TM of Prime • Reality TM of Microdata Corp.

IF YOU CAN READ THIS AD, YOU CAN TALK TO OUR COMPUTER.

(TALK IS NOT CHEAP, WHEN YOUR COMPUTER SPEAKS A DIFFERENT LANGUAGE.)



Microdata's REALITY® computer system speaks your language: ENGLISH®. So you never have to worry about odd codes and strange symbols only a computer expert could understand. In fact, anyone from the president on down can talk to the REALITY system in simple words and phrases. So you don't have to pay

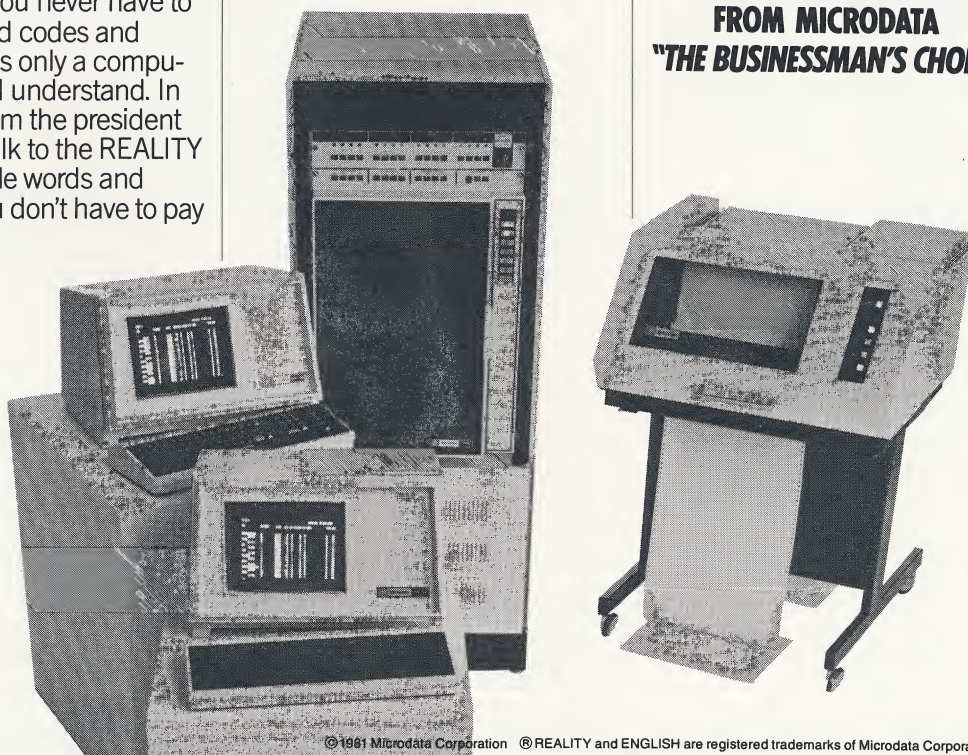
the high cost of translating data into exotic computer languages. REALITY computers can also save you money as your business grows. Because the system is modular, it grows apace with your business. Without expensive growing pains.

The REALITY computer system. For years it's been the number one choice of businessmen

actually using business computers. REALITY. We think it speaks for itself.

To learn more about REALITY, the computer that speaks your language, talk to Microdata Corporation. For complete information and a hands-on demonstration, write P.O. Box 19501, Irvine, CA 92713, or call 714/540-6730.

REALITY®
FROM MICRODATA
"THE BUSINESSMAN'S CHOICE."™



© 1981 Microdata Corporation ® REALITY and ENGLISH are registered trademarks of Microdata Corporation, Irvine, CA

PRAGMA

207 GRANADA DRIVE
APTOS, CA 95003

BULK RATE
U.S. POSTAGE
PAID
APTOS, CA 95003
Permit No. 67

ADDRESS CORRECTION REQUESTED

The Microdata Interface You Have Been Waiting For!

The **Microdata Communications Controller** is a standalone microprocessor based device that converts a microdata "dumb" port into an intelligent interface. The MCC contains buffers, logic and modem controls which are "commanded" from data basic programs.

All interaction with an MCC is in data basic programs which can activate or interrogate all modes. There are three channels, one to microdata, one for normal prism usage and one with modem controls for communication with POS terminals, time clocks, remote printers or any other RS 232 device.

Microdata to microdata communications requires only one MCC. Half duplex synchronous modems may be used for faster throughput.

Call or write for information

DYNATRONICS INC.

6200 Yarrow Dr.
Carlsbad, CA 92008

Phone: (714) 438-2511

(800) 854-1711 (CALIFORNIA)

(800) 854-6449 (OUTSIDE CALIFORNIA)

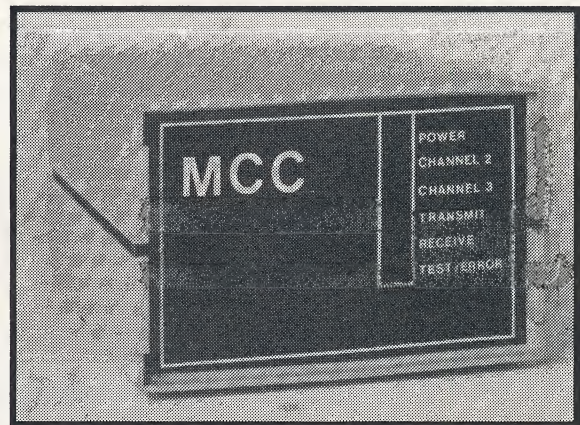
Contact: Jerry Weinert

Name _____

Company _____

Address _____
Street City State Zip

Phone () _____



(The MCC contains all solid-state circuitry and performs Power-Up self diagnostic testing.)

The MCC has more functions and capabilities than other systems ... almost always at a saving to you!

- Transparent to reality
- Dynamic configuration
- Data basic programming only
- Full modem interface RS-232
- Shares a microdata port
- Microdata to microdata
- Sync or Async
- 6 LED display
- Self test diagnostics